MASTER'S THESIS

Explaining recurrent neural networks using rule extraction

A practical evaluation

van der Wart, M.R.

Award date:
2020

EXPLAINING RECURRENT NEURAL NETWORKS USING RULE EXTRACTION

A PRACTICAL EVALUATION

by

M.R. van der Wart

in partial fulfillment of the requirements for the degree of

Master of Science
in Software Engineering

at the Open University, faculty of Management, Science and Technology
Master Software Engineering

Student number:		
Course code:	IM9906	
Supervisor:	dr. A. J. Hommersom,	Open University
Thesis committee:	dr. A. J. Hommersom,	Open University
	prof. dr. M. C. J. D. van Eekelen,	Open University

ACKNOWLEDGEMENTS

Working on a graduation assignment can be a lonely task as they say, but is never achieved alone. Therefore. I would like to express my gratitude to everyone who has supported me throughout this experience.

First of all I would like to thank my supervisors dr. A.J. Hommersom and prof. dr. M.C.J.D. van Eekelen. You have helped me to find a fitting graduation subject and your suggestions and support have been invaluable to me. Especially when I struggled with the direction and focus of my research you gently guided me back on course.

I would also like to thank my parents, who have always supported me in my technical interests and made it possible for me to study at the Technical University of Twente at the age of 17. Although personal circumstances cut this endeavor short after two and a half years, I am forever grateful for your love and support. Not reaching a master's degree then has been the primary driving force behind my current attempt to achieve it now at the age of 43. I am grateful you are still here to share this achievement with me.

Lastly, I would especially like to thank my wife Barbara and my two precious children Lorenzo and Noémi for their support. I wish I could have spent more time with you during the last years. Thank you for sticking with me during what I know was a challenging and sometimes stressful time.

All of you have been there for me, either through moral support, or practical feedback. Thank you, I could not have done this without you.

Martijn van der Wart

Contents

List of Figures

LIST OF TABLES

SUMMARY

Machine Learning (ML) has gained significant traction over the last decades and affects ever more aspects of our everyday lives. ML algorithms often perform well, but why is not always fully understood due to their increasing complexity. This lack of explainability limits the further adoption of these techniques, especially in sensitive applications, such as health care. Explainability of modern machine learning algorithms has therefore become an active research area. Recently, there have been interesting developments in the explanation of Recurrent Neural Networks (RNN) which are widely used for textual inference and speech recognition.

In this research we have explored the practical applicability of two most recent pedagogical (black-box) variants of a specific approach called, Recurrent Neural Network Rule Extraction (RNN-RE), which aims to explain the operation of RNNs in the form of automata. To this end we have applied these techniques to a RNN not specifically trained for this purpose. We explored what the practical challenges are of applying these methods to this network, looked into how each method performs and determined how well the methods contribute to gaining insight into the RNN's operation.

We have found that, although some practical aspects had to be addressed, the methods are capable of extracting automata with good fidelity. These automata, however, are too complex, either in size or in their semantics, to easily provide a global explanation of the RNN's behaviour. Therefore we conclude that, although the techniques are interesting and can certainly be useful, they are unlikely, in their current form, to provide a complete answer to the RNN explainability problem.

The main contribution of this research is to provide insight into the practical applicability of two most recent pedagogical recurrent neural network rule extraction techniques. Also we have compared these methods side-by-side under similar circumstances, which, to the best of our knowledge, has not been done before.

1

INTRODUCTION

The application of Machine Learning (ML) techniques, which are a form of Artificial Intelligence (AI), has gained significant traction over the last two decades [Mit06; JM15]. The sophistication and practical applicability of machine learning algorithms has increased significantly as both research and available computing resources have advanced [LBH15; JM15]. Nowadays machine learning algorithms are applied to many different domains and have become ubiquitous in our every day lives [LBH15; JM15]. Their application is manifold and ranges from analyzing trends and making predictions to image and speech recognition [Mit06] or supporting medical diagnostics [Kon01]. As a result more and more aspects of our every day lives are somehow influenced by machine learning algorithms.

A problem with applying machine learning techniques is that, though they perform impressively well at certain tasks, the exact reason for this is not always fully understood [SWM17]. This is especially true for Deep Learning (DL) techniques like Deep Neural Networks (DNNs). Because of their complex nested non-linear structure these networks are usually applied without knowing exactly what makes them arrive at their predictions [SWM17]. As these techniques become more and more sophisticated it becomes more and more challenging to understand and explain their operation.

This lack of explainability or transparency has hindered an even broader application of deep learning techniques, for example in areas like health care [SWM17]. As a result the explainability of modern machine learning algorithms has gained a lot of attention from the research community. Over the years many different approaches have been proposed, but particularly in recent years there have been some interesting advances on the explainability of a particular class of deep learning algorithms called Recurrent Neural Networks (RNN) [Wan+18c; Jac05]. RNNs can be used to model temporal behavior as they can capture long term dependencies in sequential data [MJ01]. This makes them particularly well suited for tasks such as language modeling [Mik12] and speech recognition [GMH13].

1.1. PROBLEM STATEMENT

Recurrent neural networks are complicated non-linear models and are usually applied in a black-box manner, meaning that there is no information provided on how they arrive

at their predictions [SWM17]. This lack of transparency hinders their application. Over the years many methods have been developed that try to extract the knowledge that is stored in these networks in a human readable form, in an attempt to achieve explainable AI [Wan+18c; Jac05]. The fast majority of these methods, however, have been studied in the context of language induction or on small artificial networks. Little or no experiments on real-life networks have been performed. It is therefore unclear if these techniques can be applied in practice and if they can indeed contribute to gain insight into the operation of recurrent neural networks.

In this research we will explore the different methods described in the literature and focus on a particular one, called Rule Extraction (RE), which attempts to extract symbolic knowledge from the RNN in the form of automata. We will use the latest techniques from a particular form of RE, called the pedagogical approach [Wan+18c; Jac05], and explore their practical feasibility by applying them to a RNN used for binary classification of a botnet, which was the result of a master's thesis by Poon [Poo18]. We will evaluate how the pedagogical rule extraction techniques perform and whether they contribute to the explainability of the RNN.

We will focus on the pedagogical approach as it treats the RNN as a black-box and makes no assumptions on its inner structure or operation [ADT95], which makes it potentially the most widely applicable RE approach.

1.2. RESEARCH CONTRIBUTION

The majority of existing rule extraction techniques are based on what is known as the decompositional approach [Wan+18c; Jac05]. This means that they use knowledge on the network's structure and require access to the inner parts of the network, hence, these are white-box approaches [ADT95]. Other techniques consider the network as a black-box. These so called pedagogical techniques use the network as an oracle to test predicted responses on input in order to build an equivalent automaton of the network [ADT95]. The number of available techniques in the latter category, however, is rather limited and most approaches are exponential in their complexity [EL06; Wan+18c; Jac05].

Despite these limitations a black-box approach is preferable, as access to the network's internal parts may not always be possible or convenient. Also, by its nature, a black-box approach makes no assumptions on the networks inner structure. As such the potential application of a black-box approach may be much broader than that of white-box approaches. In this research we will therefore primarily focus on these black-box techniques and explore their practical feasibility by applying them on a real-life RNN.

Besides exploring the practical feasibility of using pedagogical rule extraction techniques on a real-life RNN in an attempt to explain its operation, the contributions of this research are as follows:

We will perform a direct comparison of the applicability and performance of different pedagogical rule extraction techniques under similar circumstances. Such a comparative evaluation can be very informative and has not received much attention before [Jac05]. Especially aspects such as explainability and fidelity have, to the best of our knowledge, not been directly compared before.

Also extracting automaton from more complex neural networks may still be an open problem [Wan+18c; Jac05]. Therefore it is interesting to see if the studied RNN can be translated into an sufficiently equivalent automaton and which rule extraction techniques can be used to achieve this.

1.3. DOCUMENT STRUCTURE

The remainder of this document will describe our research and is structured as follows: Chapter 2 explains the research context and introduces the relevant topics. Chapter 3 presents the research design and research questions. Chapter 4 describes the evaluation metrics that will be used in our research to evaluate the performance of the rule extraction techniques and the automata they extract. Chapter 5 through 8 form the main part of our research and describe the experiments we have conducted and analyse their results. Finally, in chapter 9 we conclude on our findings and provide recommendations for further research.

2

BACKGROUND

This chapter describes the relevant concepts used in this research. The first section introduces neural networks and in particular recurrent neural networks. The second section gives a brief overview of the types of automaton used in rule extraction and explains the link between RNNs and these automaton. The third section describes rule extraction itself, the type of approaches possible and the particular techniques that will be used in this research.

2.1. ARTIFICIAL NEURAL NETWORKS

Artificial Neural Networks (ANN) are a family of machine learning models that are loosely inspired by the workings of biological brains and the way humans learn [JMM96]. In neural networks information is processed by computational units that mimic the behavior of real neurons, hence there name. These computational units consist of a set of *synapes* or weighted inputs, an *adder* which sums the input signals, and an *activation function* which limits the amplitude of the output [Hay+09]. Like its biological counterpart an artificial neuron will fire (activate) when the weighted sum of its input signal exceeds a certain threshold. The type of response is governed by the activation function. The activation function introduces a non-linearity in the response of the neuron. There are many different types of activation functions, such as sigmoid, Relu or even binary functions, but the first two are the most commonly used [Hay+09]. The output signal of a neuron is determined by the input signals, the input weights and the activation function.

Artificial neural networks combine these neurons in particular network structures to perform complex computational tasks such as image recognition. Neural networks are capable of finding patterns in data which are too complex or numerous to extract and program specifically [JMM96].

The research on these networks has a long history and started in 1943 with the work of McCulloch and Pitts [MP43]. In 1958 Frank Rosenblatt created the perceptron, an algorithm for pattern recognition [Ros58]. Combining multiple perceptrons in a layered structure led to the Multi-layer Perceptron or feed-forward ANN, which formed the foundation of modern machine learning [JMM96].

In these networks neurons are arranged in layers and they consist of an input and an output layer and one or more hidden layers [Hay+09]. The output of each layer serves as input to the next layer so that information is propagated in a forward direction through the network. Figure 2.1 presents an example of such a network with one hidden layer. Neural networks with more than two hidden layers are referred to as deep neural networks (DNN) [LBH15].

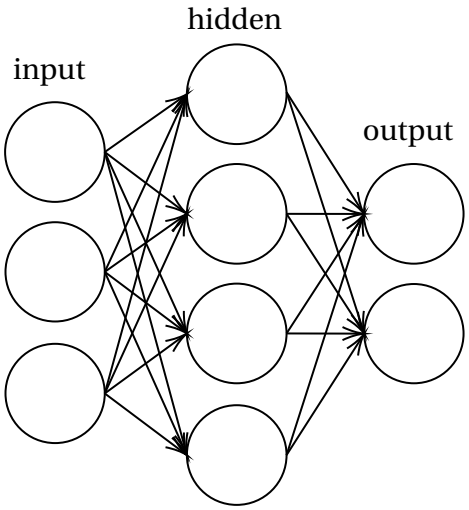


Figure 2.1: Neural network with one hidden layer

An ANN with a single hidden layer can be defined as follows:

$$h_t = f_h(W_{in}x_t) \tag{2.1}$$

$$y_t = f_o(W_{out}h_t), \tag{2.2}$$

where f_h and f_o are the hidden layer activation function and an output function, x_t and h_t are the input vector and hidden layer output vector at time t, and W_{in} and W_{out} are the weight matrices in the input and output layer.

In neural networks the hidden layers transform the input into usable features for the output layer. For example in image recognition, the hidden layers may analyse the brightness of pixels, identify edges based on brightness, recognize shapes based on lines, etc., making it possible for such network to finally determine that the image presents a cat for example.

Neural networks need to be trained and validated before they can be used. The theoretical aspects and finer details on how this is done is beyond the scope of this research as we will focus on extracting rules from existing, pre-trained, networks. Interested readers may refer to [Hay+09] or [JMM96] for more detail.

2.1.1. RECURRENT NEURAL NETWORKS

Another class of neural networks are recurrent neural networks (RNN). These networks also contain feedback connections, allowing information to travel backwards throughout the network as conceptually displayed in Figure 2.2. The neurons in these networks can not only use information from the previous layer in a feed-forward direction, but also use output of other neurons or itself in a feed-back direction [Hay+09; MJ01]. These feedback

connections effectively create a memory function which gives the network the ability of retaining information to be used at a later time. This allows the network to model temporal behaviour, making this type of ANN particularly well suited for processing sequential data [MJ01]. RNNs have been successfully used in tasks such as speech and handwriting recognition [MJ01; GMH13].

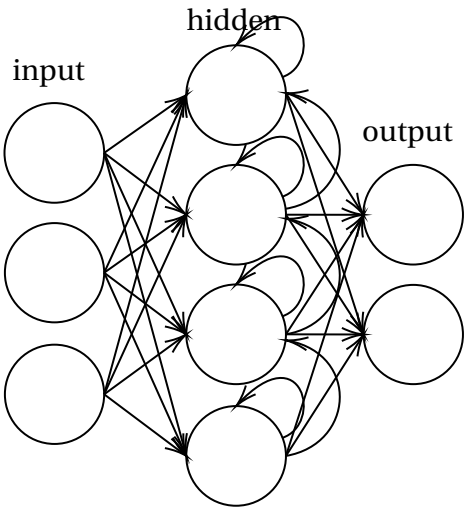


Figure 2.2: Recurrent neural network with one hidden layer

A RNN is a dynamic system as it can model behaviour over time. It can generate diverse output in response to the same input, as the response may also depend on its internal state, i.e., the state of the existing memories which represent information about previous behaviour [MJ01]. The dynamics modelled by a RNN can be continuous or discrete in time. However, the simulation of a continuous-time RNN on a digital device requires a discrete-time equivalent model. In this research we will focus on discrete-time neural networks.

With the introduction of feedback connections some practical problems arise around controlling the feedback signals. As input cycles around the network its influence on the hidden layers may blow up exponentially or decay all together [BSF+94]. Especially the later, which is know as the *the vanishing gradient problem*, is a typical problem of RNNs as it limits their ability to handle long sequences. In a way typical RNNs suffer from short-term memory.

Several different types of RNN architectures have been devised to overcome such problems. How the feedback connections are exactly implemented differs per specific RNN architecture. Also the number of hidden layers and feedback connections differ depending on the RNN's intended purpose. Commonly used RNN architectures are the Long Short-Term Memory (LSTM) and Gated Recurrent Units (GRU) architectures [Sal+18].

In this research we will focus on rule extraction techniques that treat the RNN as a black-box, therefore, we do not go into detail on these particular RNN architectures here. For more details on these networks see, for example, [MJ01; Sal+18].

More formally a RNN is a neural network that models the behaviour of a discrete-time nonlinear dynamic system. A RNN has an input x_t, an output y_t and an internal state h_t

where the subscript t represents the discrete instant of time. The dynamics are defined by:

$$h_t = f_h(x_t, h_{t-1}) \tag{2.3}$$
$$y_t = f_o(h_t), \tag{2.4}$$

where f_h is the state transition function and and f_o an output function [Pas+14; MJ01]. Or slightly reformulated to explicitly include the applied weights for each neuron:

$$h_t = f(W_{in}x_t, W_h h_{t-1}) \tag{2.5}$$
$$y_t = g(W_{out}h_t), \tag{2.6}$$

where f is a non-linear activation function and g an output function. W_{in} is the weight matrix in the input layer, W_h is the state to state recurrent weight matrix and W_{out} is the weight matrix in the output layer [SJ16]

The depth of the memory modelled within a RNN differs per RNN architecture. A simple RNN for example, may only take into account the previous values of the hidden neurons when computing its new state. It cannot 'look back' into time beyond its last state. More sophisticated models have deeper memory and can take multiple previous states into account making them better suited to handle long sequences [SJ16].

The temporal processing of RNNs is often visualized by 'unfolding' the network in time. In this view the RNN is seen as a series of 'normal' ANNs, one per time step, that all share the same weights [SJ16; SP97]. Figure 2.3 shows such a conceptual representation of a RNN using the above notation.

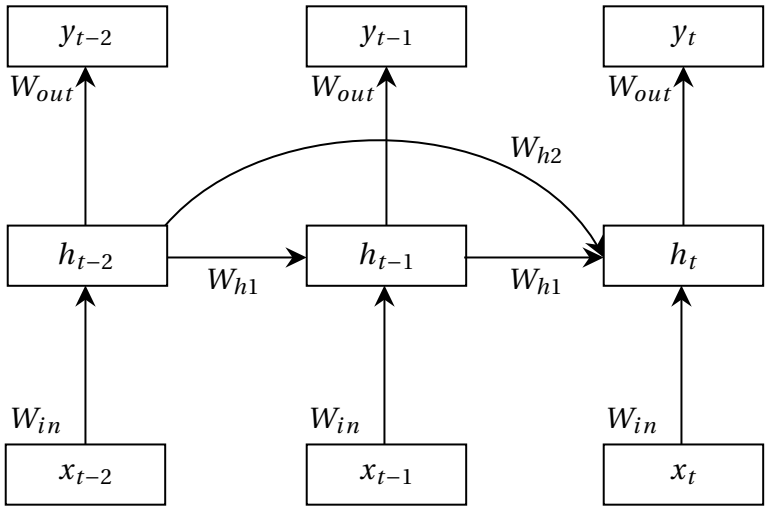


Figure 2.3: Visual representation of a RNN unfolded it in time with a depth of two time steps.

BIDIRECTIONAL RNNS

A bidirectional RNN is a combination of two separate RNNs. One RNN that moves forward through time, beginning from the start of the sequence and another RNN that moves backward through time, beginning from the end of the sequence. This makes it possible for the bidirectional RNN to compute a representation that depends on both the past and the future input [GCB16; SP97].

During this research we will evaluate the practical feasibility of black-box extraction of symbolic knowledge using a RNN trained for botnet detection. This network was the result of a master's thesis by Tho Poon [Poo18] and uses the LSTM architecture. Poon created both a uni- and bidirectional version of this RNN in his research.

2.2. AUTOMATON

The rule extraction techniques used in this research learn or extract finite-state automata (FSAs) or finite-state transducers (FSTs) from recurrent neural networks. The exact type of automaton that results from applying these techniques differs per approach. Some generate relatively simple automata, such as Deterministic Finite Automata (DFAs) [WGY18a; VO04] whilst others use richer representations, such as Weighted Finite-state Automaton (WFA) [AEG18]. In this section we will briefly discuss these different automata and explain how they relate to recurrent neural networks.

A FSA or finite-state acceptor is a finite-state machine that accepts or rejects strings of symbols from an input alphabet. A finite-state transducer (FST) is a finite-state machine that maps strings of input symbols to strings of output symbols [Hop08]. Where a finite-state automaton (FSA) defines a formal language by accepting a set of input strings a FST defines a map between sets of input and output strings, i.e., it computes a relation between the input and output language. This makes a FST a more general automaton than a FSA [Hop08; DKV09].

Both FSAs and FSTs can be deterministic or non-deterministic in their behavior. This means they will either only produce a unique sequence of state transitions (run) for each input string, or they can produce different runs for the same input string [Hop08].

2.2.1. DETERMINISTIC FINITE AUTOMATON

A Deterministic Finite-state Automata (DFAs) only produces a unique sequence of state transitions for each input string. A DFA M is formally defined by the 5-tuple [Hop08]:

$$M = \langle Q, \Sigma, \delta, q_0, F \rangle \tag{2.7}$$

where Q is a finite set of states, Σ the input alphabet, $\delta : Q \times \Sigma \rightarrow Q$ a state transition function, $q_0 \in Q$ an initial state and $F \subseteq Q$ a set of accept states.

DFA M will accept a string $w = a_1 a_2 ... _a n$ over the alphabet Σ when a sequence of states exist in Q that starts with q_0 and ends with a state in F, otherwise the automaton is said to reject the string.

The set of accepted strings form a formal language defined by DFA M. DFAs define exactly the set of regular languages which are, for example, used in lexical analysis and pattern matching. [Hop08]. Figure 2.4 depicts a DFA $M = \langle \{S_1, S_2\}, \{a, b\}, \delta, S_1, \{S_1\} \rangle$ that accepts the simple regular language $b^* + (b^* a\, b^* a)^*$.

2.2.2. WEIGHTED AUTOMATON

A Weighted Automaton (WA) or Weighted Finite State Transducer (WFST) is an automaton in which the states and transitions have weights associated to them. The weights can

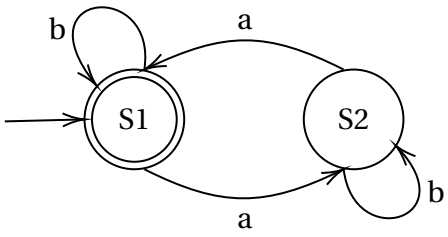


Figure 2.4: State diagram of DFA M accepting the regular language $b^* + (b^* a : b^* a)^*$

be used to model, for example, the cost of a particular path through the automaton or the probability or reliability of its successful execution. For each completed sequence of transitions the individual weights are aggregated in an appropriate manner. The behaviour of weighted automata can thus be seen as associating to each input string the total weight of its execution as defined by this aggregation function [DKV09].

As an example we can take an automaton in which the transitions have as weights the amount of energy needed for their execution. The total amount of energy required for a given path in this weighted automaton is then simply the sum of the weights of the individual transitions. The automaton could be used to find the minimal amount of energy required to execute a certain input string, i.e. the optimum path out of the successful paths realizing the input string. If, in another case, the weights would represent probabilities, the likelihood or reliability of a path could be defined as the product of the probabilities of its transitions.

A weighted automaton M over a set of weights S is formally defined by the 5-tuple [DG07]:

$$M = \langle Q, \Sigma, \lambda, \mu, \gamma \rangle \tag{2.8}$$

where Q is a finite set of states, Σ the input alphabet, $\mu : Q \times \Sigma \times Q \to S$ the *transition-weight function*, and $\lambda, \gamma : Q \to S$ the *initial-weight* and *final-weight functions* which are weight functions for entering and leaving a state, respectively [DG07].

To make operations on the weighted automaton well-defined the set of weights S is often required to form a semiring, which is an algebraic structure that specifies the binary operations of addition and multiplication on a set and defines certain constraints over their operation. More detailed information on weighted automata and the related algebra may be found in [DG07; DKV09].

Figure 2.5 depicts a WA that accepts the same simple regular language $b^* + (b^* a\, b^* a)^*$, but now also models, for example, the likelihood of individual strings in that language. Each transition carries the probability of encountering a certain input character and the product of all consecutive transitions in a run represents the likelihood of an input string. This particular WA would deem short input strings more likely than longer strings as the recursive transitions have a low probability.

LINEAR REPRESENTATION

A WA can also be represented in a linear fashion using vectors and matrices [Bal+14]:

$$A = \langle \alpha_0, (M_\sigma)_{\sigma \in \Sigma}, \alpha_\infty \rangle \tag{2.9}$$

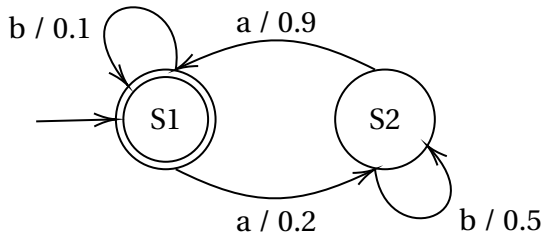


Figure 2.5: State diagram of WA M accepting the regular language $b^* + (b^* a\ b^* a)^*$ and modelling the likelihood of strings in this language.

where the vectors α_0 and α_∞ provide the initial and final weights (i.e. the values of functions λ and γ for each state) and each matrix M_σ corresponds to the σ-labeled transitions weights (i.e. $M_\sigma(q_1, q_2) = p \iff \mu(q_1, \sigma, q_2) = p$)

This linear representation is used in one of the rule extraction methods that we will explore in our research.

2.2.3. CONNECTION BETWEEN RNN AND AUTOMATA

The relationship between RNN and finite-state automata has been studied in length over the last decades and many studies have looked at how RNNs can be used to represent certain finite state machines [Gil+92; HH94; CSM89]. In a way this is the exact opposite of what we try to achieve in this study, however, the described relationship between the two domains is still relevant.

As described earlier, a recurrent neural network is a dynamic system and as such it is composed of two parts, its state and its dynamics [MJ01]. In a discrete-time RNN the output values of the recurrent neurons are stored in the "memory" of the RNN. How these values are stored and for which neurons, depends on the particular RNN architecture. These stored values form the state of the RNN and summarize all the information about past behaviour necessary to determine its future behaviour [MJ01]. The RNN's state space is formed by all its possible states. The dynamic of the RNN, which we will assume to be deterministic [MJ01], describes how the states evolve over time. Each recurrent neuron computes its next output value based on the current state and input of the RNN. These new values form the new state of the RNN. From this it is clear to see that a discrete-time RNN models a state machine or automaton: The RNN resides in a certain state and depending on its current input it transitions to a new state.

The type of application of the observed RNN determines the appropriate type of automaton that might be used to represent it. A RNN used for binary classification might be represented by a FSA as it is basically a form of acceptor, i.e., it either classifies some input as positive (accepts it) or as negative (rejects it). A RNN used for a prediction problem, outputs a real or continuous value and could potentially be represented by FST.

Whether a particular RNN can be sufficiently represented by a finite-state automaton is not a given in general. However, it has been proven that DFAs can be successfully extracted from RNNs that have been trained to recognize regular languages [Wan+18c]. If this is also possible for RNNs outside this domain is an open question and part of this research as we

will investigate if we can successfully extract sufficiently equivalent automata from a RNN trained for botnet detection.

2.3. RULE EXTRACTION

As mentioned in the introduction, it is, in most cases, difficult or near impossible to inspect, analyze and verify the knowledge captured in trained artificial neural networks, and in our case recurrent neural networks [Wan+18c; Jac05]. Over the last decades many attempts have been made to address this issue resulting in many different approaches. These approaches include techniques such as Taylor decomposition, relevance propagation, sensitivity analysis and information based techniques that try to 'visualize' the learned knowledge in the neural network as a result to input stimuli [MSM18; Mon+17; Arr+17]. These methods provide local explainability, since they focus on explaining how the network comes to a decision on a given datum [Gui+18].

Another approach, which has seen a recent revival for RNNs, is to investigate how symbolic knowledge might be extracted from trained neural networks, since this type of information is usually regarded as easier to understand [Wan+18c]. This technique, called Rule Extraction (RE), tries to provide a general explanation of the network [Gui+18] by extracting a set of symbolic rules that describe the overall behavior of the network. Depending on the extraction technique and type of neural network, these rules may take the form of a decision tree, a set of propositional logic formulae or some other form of symbolic representation [ADT95; EL06; KM09].

In the case of RNNs the symbolic knowledge captured by a trained RNN is extracted in the form of a finite-state automaton [Jac05; Wan+18c]. It is this technique, often referred to as RNN-RE [Jac05], that is the subject of the this research. In the case of RNN-RE the term rule should be interpreted quite loosely and be seen more as a synonym for the information expressed in such an automaton.

The motivation behind RNN rule extraction is that the information processing of an RNN can be treated as *"a mechanism for representing knowledge in symbolic form where a set of rules that govern transitions between symbolic representations are learned"* [Wan+18c]. If viewed this way we can think of a RNN as an automated reasoning process with production rules, which should make it easier to understand [Wan+18c]. Rule extraction for recurrent neural networks is then essentially the process of finding rules that sufficiently approximate the behaviour of the RNN [Jac05].

The most commonly used RNN rule extraction approach is to extract DFAs from recurrent networks, mostly on networks that are trained for grammatical induction. Although there are techniques that can successfully extract DFAs from these relatively simple recurrent neural networks it is still unclear how these techniques perform on more sophisticated networks, such as LSTMs and GRUs. Also the type of activation function used in the networks and the data used to train them might influence the extraction process and quality of the resulting automata [Jac05; Wan+18c]. It is one of our research objectives to investigate the practical applicability of rule extraction techniques on a RNN with the LSTM architecture.

2.3.1. RULE EXTRACTION APPROACHES

Individual rule extraction methods for recurrent neural networks can be placed into the following taxonomy [Wan+18c; CBP19; Jac05; Wan+18a]:

1. **Compositional Approaches**
   Techniques in which rules are constructed based on ensembles of hidden neurons, i.e., the hidden layers. These techniques directly access the internal state of the RNN.

2. **Decompositional Approaches**
   Techniques in which rules are constructed based on individual hidden neurons. These techniques also require access to the internal state of the RNN.

3. **Pedagogical Approaches**
   Techniques that construct rules whilst regarding the RNN as a black-box and that do not access the inner state of the RNN.

4. **Eclectic Approaches**
   Techniques that combine decompositional and pedagogical approaches.

The majority of existing rule extraction techniques are decompositional white-box approaches [Wan+18c; Jac05] which means they employ knowledge on how the network is constructed, i.e., its architecture, and require access to the internal parts of the network [ADT95]. Other techniques treat the network as a black-box and do not use the network's internals. These pedagogical approaches use the network as an oracle to test predicted responses to input and use this information to construct an equivalent automaton [ADT95].

The number of pedagogical techniques is limited and most are exponential in their complexity [EL06]. Despite these limitations pedagogical approaches are preferable since access to the internal parts of a network may not always be possible or desirable. In this research we will therefore primarily focus on these black-box pedagogical techniques.

2.3.2. PEDAGOGICAL RULE EXTRACTION

As mentioned, the number of pedagogical RNN rule extraction techniques described in the literature is rather limited. During an extensive literature study on the subject we have been able to find only three such methods of which one could be considered to be only partially pedagogical. Two of the methods have been published only in the last two years, showing that this is still a very active area of research. To the best of our knowledge these three methods are the only pedagogical techniques available to date.

The pedagogical rule extraction techniques for recurrent neural networks that we have found are:

- ***Explaining Black Boxes on Sequential Data using Weighted Automata*** by Ayache et al. [AEG18].
  Uses a spectral learning algorithm to extract WAs from real valued RNNs.

- ***Extracting Automata from Recurrent Neural Networks Using Queries and Counterexamples*** by Weis et al. [WGY18a].

Uses an exact learning approach based on the L*-algorithm [Ang87] for learning DFAs from black-boxes.

- ***A Machine Learning Method for Extracting Symbolic Knowledge from Recurrent Neural Networks*** by Vahed et al. [VO04].
  Uses a symbolic learning algorithm to extract DFAs by observing the output of trained RNNs for all strings up to a given string length.

Two of the three techniques extract DFAs and one extracts Weighted Automata (WA), which are a richer, more expressive type of automata [DG07]. For our research we will use one technique from both varieties to increase the change of finding a sufficiently equivalent representation for our case study RNN and to broaden our comparative evaluation of the different available pedagogical rule extraction techniques.

Additionally we will focus on the first two techniques as they are the latest additions to the field and thus could be considered the state-of-the-art in pedagogical RNN rule extraction. In the remainder of this section we will discuss these pedagogical rule extraction techniques in more detail.

EXPLAINING BLACK BOXES ON SEQUENTIAL DATA USING WEIGHTED AUTOMATA

One of the most recent methods for extracting symbolic knowledge from a RNN in a black-box manner is the work of Ayache et al. [AEG18]. This method extracts Weighted Automata (WA) from recurrent neural networks trained for prediction tasks, i.e., networks that assign numerical values to symbolic data. In this sense the method is unique, as it is, to the best of our knowledge, the only method that does not specifically target RNNs trained for binary classification.

The method treats the RNN as a pure black-box and does not require access to its internal state. The symbolic model, i.e., the weighted automaton, is obtained solely by using the trained RNN as an oracle, feeding it inputs and observing its output. The method is not restricted purely to RNNs, but aims at "extracting a finite state model from *any* black-box that computes a real valued function on sequential symbolic data" [AEG18].

The method proposed by Ayache et al. aims at providing global interpretability of such black-box models. This means that it attempts to provide a general explanation of the behaviour of these black-box models and not just attempts to explain how a decision is taken on a particular datum [Gui+18].

The core of the method is based on a spectral learning algorithm that allows to extract a WA from such black-boxes. This algorithm was used in previous research [BDR09; HKZ12; BM18] where the WAs where extracted using estimations based on counting the occurrences of strings in a learning sample. In contrast to these methods the method proposed by Ayache et al. uses the trained black-box model itself to achieve this goal.

The spectral learning algorithm is based on the fact that functions that assign real values to strings: $f : \Sigma^* \to \mathbb{R}$, which are known as rational series [Sak09], can be defined by a WA under certain conditions of finitude. Rational series can be associated with a bi-infinite matrix: $H \in \mathbb{R}^{\Sigma^* \times \Sigma^*}$ called a Hankel matrix [Bal+14] whose entries are defined as $H(u, v) = f(uv), \forall u, v \in \Sigma^*$. The Hankel matrix is thus a matrix representation of the rational series in which the rows are string prefixes and the columns string suffixes. The di-

mension of the vector space of a matrix is defined by its *rank*. It is the maximum number of linearly independent column (or row) vectors in the matrix and a measure for the 'true' informational value of the matrix [Bou98]. A rational series can be defined by a WA iff the rank of its Hankel matrix is finite [CP71]. The intuition here is that when the number of unique strings defining the rational series (i.e. the rank of its Hankel matrix) is finite, it can be represented by an automaton with a finite number of states.

The above theorem can be used to generate a WA from its Hankel matrix H. Particularly, the WA can be constructed using special finite sub-blocks of this matrix. If the sub-block H_B of H has the same rank as H and is prefix-close, which means that for each row value, all prefixes of that value are also rows of H_B, it can be used to construct the WA. The intuition here is that when the sub-block contains the same informational value as the Hankel matrix, it can be used in its place for the construction of the WA.

The algorithm proposed by by Ayache et al. attempts to compute a WA from such finite sub-blocks of the Hankel Matrix that have been established by querying the black-box model. It starts by building the sub-block H_B based on random sampling, either using a dataset or the uniform distribution on symbols and a maximum length parameter. If a string is selected using this sampling approach it is added to the rows of H_B, together with all its prefixes, to be prefix-close. The same applies to its suffixes, which are added as columns to H_B. This is repeated until the row and column vectors of the matrix reach a predefined size. Next, the sub-block is filled with values obtained from querying the black-box model. Finally the algorithm extracts a linear representation of a minimal WA from the Hankel matrix sub-block using a particular linear algebra called rank factorization.

The authors evaluate their approach on a RNN architecture based on two GRU layers which has been trained on several datasets. The authors compare the similarity between the probability distribution of samples taken from the RNN and the extracted WA. They additionally compare the proximity of the RNN and WA models on the task of predicting the next symbol in a sequence. They perform these experiments using different settings of the hyper-parameters: size p and s of the algorithm, which are the size of the row and column vectors of H_B.

Based on these experiments they conclude that their method allows good approximation of black-boxes and achieves a great approximation of the black-box model even for low values of the hyper-parameters. The authors do, however, note that they did not chose the most favorable method for generating H_B. Using other methods than sampling based on the assumption of uniform distribution on symbols may lead to better approximation. Also choosing a larger basis could allow better results. These are areas that we will explore further in this research. We will, for example, evaluate if using sampling from the training dataset of the RNN is favorable over assuming the uniform distribution on symbols. Also we will vary the applied length parameter and size of H_B.

Finally, the authors discuss the interpretability of WAs. They note that WAs can be hard to read due to their non-deterministic nature, especially when the number of states increases. As an answer the authors mention their method depends on the rank parameter, which can be tweaked, and that their method already produces approximations of acceptable quality for small rank values. This allows users to prefer readability over performance and chose a small rank value to obtain a small WA. This is an interesting point that we will further

investigate in this research. We will apply the algorithm using different rank values and evaluate the quality and interpretability of the produced WAs.

On important point to note is that the method proposed by Ayache et al., as mentioned, works on real valued black-box models. In our research we will, however, evaluate rule extraction on a RNN trained for binary classification. How the method of Ayache et al. can be applied to such a network and what effect this has on the performance of the extraction algorithm is not immediately apparent and will be part of our research.

EXTRACTING AUTOMATA FROM RECURRENT NEURAL NETWORKS USING QUERIES AND COUNTEREXAMPLES

Another recent method for black-box rule extracting from trained RNNs is the work of Weiss et al. [WGY18a]. This method extracts Deterministic Finite Automata (DFA) from recurrent neural networks trained for binary classification tasks. The method uses an exact learning approach and abstraction method in which the RNN is used as an oracle to obtain a DFA that sufficiently describes the state dynamics of the trained RNN. The method proposed by Weiss et al. aims at providing global interpretability of the black-box model.

Exact learning states that a concept can be precisely learned from what is known as a *minimally adequate teacher* [GK95]. This is an oracle that is capable of answering two types of queries: *membership queries* and *equivalence queries* [GK95]. Membership queries, as their name implies, state if a given string is part of the model, i.e., if it is accepted or rejected. Equivalence queries state if a given hypothesis (model) is equal to the concept held by the teacher. If not, an example is returned on which the concepts disagree (a *counterexample*).

In their method Weiss et al. use the L* exact learning algorithm [Ang87] which is based on this principle. The L* algorithm allows to extract DFAs from a minimally adequate teacher. Weiss et al. use the RNN is as the teacher for this algorithm. Membership queries can be directly answered by the RRN, but the main challenge is answering the equivalence queries.

The core of the method of Weiss et al. is how the equivalence queries are answered during the exact learning procedure. To this end they use a finite abstraction of the RNN R as a hypothesis of the ground truth of R. This abstraction is refined during learning. The DFA A being learned by the L* algorithm forms another hypothesis of R. The key idea is that the two hypotheses must at least be equal to each other to be equal to R. Whenever the two hypotheses disagree on a sample during learning, its true classification can be found in R. This will either lead to a counterexample to A, when the finite abstraction and R agree, or to a refinement to the abstraction in the other case.

The finite abstraction of the RNN R is created using a state partitioning technique from Omlin and Giles [OG96]. This methods partitions the RNN's internal states (the microstates) into equally sized hypercubes (the macrostates) and conducts a breadth-first search (BFS) by feeding the network input patterns until no new partitions are visited [Jac05]. The search begins in the network's initial state and continues according to the network's transition function. The transitions among the macrostates (induced by input patterns) form the basis for the extracted DFA [Jac05]. The method extracts a DFA for which every state is a partition, and the state transitions and classifications are defined by a single sample from each partition.

The abstraction of the RNN R is refined during learning by finding a new state space

partitioning that increases the partitioning with exactly one state, in such a way that the abstraction and R now agree on all states seen so far. Weiss et al. use a state clustering technique based on a support vector machine (SVM) with a non-linear radial basis function (RBF) kernel to separate the conflicting state from the RNN's state space in order to find the new state space partitioning.

The key intuition to the efficiency of the approach of Weiss et al. is that, as the L* algorithm always proposes a minimal DFA in equivalence queries, each state in the abstraction and learned DFA must be equal with respect to classification and transitions if both DFAs are found the be equal. As the extraction of the finite abstraction is effectively a BFS traversal of itself this allows the states of both DFAs to be associated during this extraction. This makes it possible to perform a parallel exploration of both DFAs, significantly speeding up the algorithm.

The authors evaluate their approach on a RNN architecture based on two GRU layers which has been trained on the Tomita grammars [Tom82], which have been widely used in the rule extraction literature, and on some substantially more complicated languages. They additionally also evaluate their approach on some LSTM networks. The authors asses the accuracy of the extracted DFAs by comparing them against their networks on their train sets and on a set of 1000 random samples of different word-lengths. Additionally they also compare their method against a direct application of the method from Omlin and Giles [OG96] and against using random sampling for counter example generation.

Based on these experiments they conclude that their method is able to find small and accurate DFAs representing a given RNN, when such DFAs are available. They also conclude that their method does this in a fraction of the time required by other methods to complete their extractions. Also their method, unlike other extraction approaches, works with little to no parameter tuning, and requires very little prior information to get started.

In their evaluation the authors also mention a limitation of their method in returning DFAs for networks with more complicated behavior. Due to L*'s polynomial complexity and intolerance to noise, the extraction becomes extremely slow and returns large DFAs for such networks. The authors have found that their method builds a large DFA, and times out during refinement when applied to an RNN that has failed to generalize properly to its target language. This last point indicates a potential problem with applying this method to practical networks. We will further investigate this point as part of our research.

One important point to note is that the method proposed by Weiss et al. uses a state space clustering approach to generate and refine the abstraction of the RNN. This method uses information on the state space of the RNN and thus requires access to the network's internal state space representation. This means that this part of the their method is in fact a white-box approach, rather than a black-box approach [Jac05]. So ultimately the method proposed by Weiss et al. could indeed be seen as being only partially pedagogical.

The authors state in their conclusion: "As our method makes no assumptions as to the internal configuration of the network, it is easily applicable to any RNN architecture, including the popular LSTM and GRU models." [WGY18a]. However, as their method employs white-box techniques, as mentioned, this may prove not be entirely accurate and its practical applicability on such networks remains uncertain. How the method of Weiss et al. can be applied to such a network will be further explored as part of our research.

3

Research Design

3.1. Research Objective

In order to gain insight into the complex operation of trained recurrent neural networks, several techniques have been developed over the years that attempt to extract symbolic knowledge from these networks in the form of finite state automata [Jac05; Wan+18c].

Many of these techniques, however, assume a certain network architecture or require access to the network's internal components, rather limiting their application [Jac05; Wan+18c]. More recently, however, techniques have emerged that treat the trained RNN as a black-box and make no assumptions on its internal operation [Wan+18c; AEG18; WGY18a]. These so called, pedagogical or black-box techniques, may therefore have great potential to be used on a wider range of networks and may thus help to push the field of explainable AI forward.

The fast majority of rule extraction techniques, however, have only been evaluated on RNNs trained to recognize regular languages or on toy examples. Very few, if any, rule extraction techniques have been applied to practical RNNs. Also there has been, to the best of our knowledge, no direct comparison between black-box rule extraction techniques under similar circumstances so it is unclear how the different techniques compare.

It is the objective of the this research to gain insight into the practical feasibility of the different black-box rule extraction techniques and achieve a comparative evaluation of their performance, both in quantitative terms based on quality aspects, such as extraction time and truthfulness, as well as in qualitative terms based on the interpretability and explanatory qualities of the extracted rules.

To this purpose we will explore how the state-of-the-art in black-box rule extraction techniques can be applied to a practical RNN that has not been specifically trained for this purpose, how well they perform and if they can ultimately contribute to gain insight into the RNNs operation.

3.2. Research Questions

During the course of this research we want to answer two main questions:

RQ1 Can the black-box rule extraction techniques be applied to a practical RNN not specifically trained for this purpose?

RQ2 Do the extracted rules provide insight into the RNN's operation?

To do this we will break down the main question into the following sub-questions:

RQ1.1 What is the best way to evaluate the overall extraction performance and the quality of approximation of the extracted automata?

RQ1.2 How can the case study RNN, with its continuous input, be used by the black-box rule extraction techniques that expect symbolic data?

RQ1.3 Can the WA rule extraction method, aimed at models that compute a real valued function on sequential symbolic data, be used for binary classification?

RQ1.4 Can the techniques be successfully applied to the case study RNN, what are the best extraction settings for each technique and how well can each technique approximate the RNN's behavior?

RQ2.1 How should the numerical semantics of the extracted WAs be interpreted, particularly in a binary classification scheme?

RQ2.2 How interpretable are the extracted automata and how well does each technique help to explain the RNN's operation?

3.3. RESEARCH METHOD

This research will use the exploratory and empirical research methods. The overall research uses an exploratory approach to gain insight into how the the different black-box rule extraction techniques can be applied in practice and for determining how the quality and explainability aspects of rule extraction may be best evaluated. Individual steps in this research will use an empirical approach to compare (aspects of) the different techniques and find the best settings for each technique. This means that the research will be both qualitative and quantitative in nature.

The objective of RNN rule extraction is to extract an automaton that gives insight into the RNN's operation. Clearly, the extracted automaton must be of sufficient quality to achieve this. What this quality is, however, and how we can measure it, first needs to be established. Also, we want to evaluate the practical applicability of the different RNN rule extraction techniques. To do this we must define what aspects of their operation and performance determine this. In chapter 4 we will establish criteria and metrics for these topics based on a study of the rule extraction literate and our extensions of these subjects.

We will evaluate the practical applicability of the different RNN rule extraction techniques by evaluating their performance on a case study RNN. This RNN is trained to detect botnet traffic and is the result of the master's thesis from Tho Poon [Poo18]. The case study RNN and related datasets are described further in chapter 5.

To answer the research questions in more detail we address the practical challenges of applying the rule extraction techniques to the case study RNN, such as converting its continuous input into symbolic data. We will explore different possible approaches and perform experiments to establish to most applicable solutions. As our case study RNN is a binary classification model and the WA rule extraction method is primarily aimed at regression type problems, we need to pay special attention to this aspect of applying this technique to our case study RNN. We will explore how we can use the WA rule extraction method on binary models and how to interpret the resulting WAs in a binary classification scheme. Chapter 6 and 7 describe these topics in more detail.

In chapter 8 we will use the results from these previous explorations to apply the rule extraction techniques to the case study RNN. We will evaluate their performance using the criteria we established and subjectively evaluate the interpretability of the extracted automata in order to answer our research questions.

Finally, we will present our conclusions and provide recommendations for future work in chapter 9.

4

Extraction Evaluation Metrics

As we want to evaluate the extraction performance of the different pedagogical rule extraction techniques, we need to establish measures to perform such an evaluation. To this end we will look into the criteria often applied in the rule extraction literature, such as rule quality and algorithmic complexity [ADT95; Jac05; Wan+18c; FM15] as well as those mentioned in recommendations for future work. Particularly we will use aspects from the approach taken in [Wan+18a], where the extracted automata are compared to the original RNN in terms of accuracy and fidelity or truthfulness.

In this chapter we will describe the different metrics that we have derived and will apply during our research.

4.1. Quality of Approximation

One of the aspects we want to evaluate is the quality of approximation of the extracted automata. With this we mean the fidelity or truthfulness of the extracted automata with regards to the trained RNN from which they where extracted, or, in other words, how closely the extracted automata model the RNN's behavior. This, we feel, is an important aspect of rule extraction as we want to gain insight into the *true* behavior of the RNN. In the rule extraction literature this aspect is captured as part of a larger concept called *rule quality* [ADT95; Jac05].

In this research we are evaluating the extraction performance on RNNs trained for binary classification. Binary classification is the task of labeling certain input as belonging to either the positive or negative case. For our case study RNN this translates into classifying network traffic as either botnet related or not. Since we are using a binary classification RNN, the extracted automata should also model a binary classifier. Essentially then the evaluation of how closely an extracted automaton models the trained RNN's behavior can be seen as comparing two different binary classifiers. Therefore, we will first look into metrics and techniques that measure different aspects of binary classification performance and that we will need throughout our research. Subsequently we will use these metrics to define a fidelity metric which we will use to determine how closely the extracted automata model the RNN's behavior.

4.1.1. BINARY CLASSIFICATION METRICS

In the field of machine learning it is standard practice to evaluate the classification performance of a binary classifier against a given dataset and is often used to compare different classification models or to asses parameter settings [Pow11].

To measure the performance of a binary classifier several metrics can be used [Pow11]. They are all based on comparing the classifier under study against a known ground truth on a given dataset and revolve around four key concepts: True Positive (TP), True Negative (TN), False Positive (FP) and False Negative (FN). TP and TN are the number of samples the classifier correctly identified as positive and negative cases respectively. FP and FN are the number of negative samples that where incorrectly classified as positive and vice versa.

The most well-known and straightforward metric is *accuracy* which is defined as:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{4.1}$$

Accuracy takes all samples into account and calculates the percentage of correct classification. By its definition accuracy gives equal weight to both the positive and negative classes. This makes this metric sensitive to uneven class distribution [Pow11]. This means that the metric can be affected by how evenly balanced the positive and negative samples are throughout the data. Accuracy can hide poor classifier performance when the classes are highly skewed. Take, for example, a dataset where only 2% of all samples belong to the positive class and a binary classifier that always returns the negative class label. Clearly the classifier does constitute a sensible model, but its accuracy would still be reported as 98%.

In a lot of practical situations unbalanced data may be the case. Quite often the positive class is the minority class, as is likely the case for our botnet detection RNN, which could render the accuracy metric unreliable. To address the issue of class imbalance two other metrics *precision* and *recall* are often used to measure the performance of a classifier.

$$Precision = \frac{TP}{TP + FP} \tag{4.2}$$

$$Recall = \frac{TP}{TP + FN} \tag{4.3}$$

Precision measures the percentage of correctly identified positive samples and expresses the classifier's ability to avoid false positives. Recall or sensitivity measures the percentage of positive samples that was identified by the classifier and expresses the classifiers ability to detect all positive cases. Both precision and recall focus on the positive samples and are thus less affected when the positive class is the minority. Both metrics would report a score of 0% for the aforementioned classifier, clearly indicating its poor performance.

To express a classifier's performance in a single value, the *F1-score* is often used, which is the harmonic mean of precision and recall.

$$F1 - score = 2 \times \frac{precision \times recall}{precision + recall} \tag{4.4}$$

As mentioned, by default precision and recall, and thus the F1-score, have a bias towards the positive class. This may distract from poor performance on the negative class. For this reason precision, recall and F1-score are often reported separately for both the positive and negative class [Pow11]. We will also do this when we report performance figures in our experiments.

4.1.2. ROC ANALYSIS

As mentioned, the case study RNN is a binary classifier and therefore the extracted automata should also model a binary classifier. The WA extraction method we will evaluate, however, extracts automata that output a continuous value rather than a binary value. We thus have to convert their output to a binary classification by applying a threshold value.

To evaluate the impact of this WA valuation threshold and to determine its optimal value we will use a technique called receiver operating characteristic (ROC) analysis. In a ROC analysis the continuous output value of the classifier is binarized using different threshold values and compared to the actual classification labels. The ROC-curve plots the true positive rate (TPR) of a binary classifier against its false positive rate (FPR) at various threshold settings. It gives insight into the binary classifier's performance as its discrimination threshold is varied [Pow11]. For the optimal binary classifier we want a high TPR and a low FPR. This means that the best performing threshold will be found in the top left corner of the ROC-curve [Pow11].

4.1.3. FIDELITY METRIC

Although there seems to be no real consensus on the exact definition of a fidelity or truthfulness measure in the rule extraction literature, we will use aspects from the approach taken in [Wan+18a] to measure how closely the extracted automaton models the RNN's behaviour. In this study the authors investigate the verification of RNN's using DFA extraction to asses their vulnerability against adversarial input. To evaluate the performance of the DFA extraction they define, amongst others, a fidelity metric for measuring the quality of the extracted DFA's. However, their metric is based on accuracy which is, as mentioned in the previous section, sensitive to class imbalance. Building on [Wan+18a] and the previous definitions we define our own extraction fidelity metric, which is based on precision, recall and F1-score.

Normally, these precision, recall and F1-score are determined on a particular dataset. As we would like to asses the extracted automata against the RNN, but also against another ground truth model, such as the RNN's training or test dataset, we redefine these metrics with a variable ground truth.

Given a model m and a dataset $B = \{X, Y\}$ consisting of samples x and labels y_x. Let X_i^m denote the set of samples classified by m as having the label i, i.e., $X_i^m = \{x \in X | m(x) = i\}$. Then X^m can be decomposed into disjoint subsets X_i^m. Let $i = 0$ be the negative label and $i = 1$ the positive label. Given a ground truth model g with the same semantics as m we have the following metrics for evaluating the performance of RNN rule extraction.

Precision and recall. The precision $Prec_i(m, g, X)$ and recall $Rec_i(m, g, X)$ for class i of model m on data set X against ground truth g are defined as:

$$Prec_i(m,g,X) = \frac{X_i^m \cap X_i^g}{X_i^m \cap X_i^g + X_i^m \cap X_{1-i}^g} \tag{4.5}$$

$$Rec_i(m,g,X) = \frac{X_i^m \cap X_i^g}{X_i^m \cap X_i^g + X_{1-i}^m \cap X_i^g} \tag{4.6}$$

Fidelity. The fidelity $Fid_i(m,g,X)$ for class i of model m on data set X against ground truth g is the F1-score of that class against the ground truth and defined as:

$$Fid_i(m,g,X) = 2 \times \frac{Prec_i(m,g,X) \times Rec_i(m,g,X)}{Prec_i(m,g,X) + Rec_i(m,g,X)} \tag{4.7}$$

To asses the fidelity for both classes in one metric we combine the individual class fidelities in two ways. The first metric we will use is the weighted average of the individual class fidelities, which we will refer to as the average fildelity (4.8). This metric gives a good overall impression of the model's fidelity with respect to the ground truth. The second, more strict, metric we will use, is the minimum of the individual class fidelities, which we will refer to as the minimum fildelity (4.9). This last metric will highlight any problems the model may have with a particular class.

$$Fid_{avg}(m,g,X) = \frac{Fid_0(m,g,X) + Fid_1(m,g,X)}{2} \tag{4.8}$$

$$Fid_{min}(m,g,X) = min(Fid_0(m,g,X), Fid_1(m,g,X)) \tag{4.9}$$

4.2. COMPREHENSIBILITY OR INTERPRETABILITY

Another important aspect of rule extraction performance we want to evaluate, is how well the extracted automata convey the trained RNN's semantics, i.e., how well they give insight into the RNN's behavior. This aspect is of a more subjective nature and has bearing on other research topics such as data visualization and ergonomics. In the rule extraction literature this aspect is another important element of the larger *rule quality* concept [ADT95; Jac05].

In order to contribute to understanding the operation of the RNN, the extracted automata themselves need to be understandable. Clearly this entails many aspects and relates to other research topics such as cognitive sciences, psychology, ergonomics and others. However, as our research scope is limited we restrict this aspect of our research to a basic subjective evaluation only.

In order to gain understanding of an unknown phenomena or domain, one needs to be able to build a mental or conceptual model of it [Joh80]. Clearly the amount of (visual) information that needs to be processed plays an important role in this process. We feel that the size of the extracted automata plays an important role in this respect as it directly influences their overall interpretability. The size of an automaton is determined by the number of states as well as the number of transitions and can thus be quantified. Furthermore, the number of symbols in the alphabet used by the automata will also be a factor as transitions

may be harder to interpret when they contain multiple symbols. This number can also be quantified.

We will compare the pedagogical rule extraction methods based on these quantities, however, we feel the semantics of the automaton will also greatly influence interpretability. The more complicated the semantics of the presented information, the more difficult it will be to interpret its meaning. A DFA has very simple, straightforward semantics whereas a WA has more complicated semantics, which may negatively affect its interpretability. Obviously this is a difficult aspect to quantify. We will again, limit ourselves to a subjective evaluation.

4.3. TRANSLUCENCY OR GENERALITY

In [Jac05] *translucency*, or the degree to which the rule-extraction algorithm 'looks' inside the RNN, is mentioned as a distinguishing feature. In [FM15] a similar aspect is captured under *generality*, or the extent to which a method requires special training regimes or restrictions on the model architecture. Both express how generally applicable the extraction method is to different models and training setups.

The less assumptions an extraction method places on the model under study, the more generally applicable it is. A pedagogical approach, in principle, requires no internal knowledge or structure and can therefore be applied to other black-box models as well. The WA method is purely pedagogical or black-box, so it seems to score well in this category. The DFA method on the other hand, requires access to the internal state of the RNN to form the hypothesis on which the L* algorithm works, making it, as mentioned, not a purely black-box approach. How this aspect impact the practical application of each method will be further investigated in our research.

Another associated quality aspect is mentioned in [Jac05] called *portability*. This describes how well an RE technique covers the set of available RNN architectures. As we will investigate only one type of RNN architecture, the LSTM architecture used by our case study RNN, we will not cover this aspect in our research.

4.4. ALGORITHMIC COMPLEXITY

The algorithmic complexity of a method is also mentioned in the rule extraction literature as a possible measure for comparing different extraction methods [FM15]. This aspect refers to the overall efficiency of the algorithm. Some methods are so complex that they are only applicable to very small toy examples, while other methods perform well even on large 'real life' problems. However, as mentioned in [Jac05], the algorithmic complexity of RE algorithms is unfortunately often an open question as authors seldom analyse this explicitly. Especially for RNN-RE the complexity issue has not received much attention [Jac05]. It is in itself quite complex to measure the expected time and space requirements as they are influenced by many factors, such as number of input symbols, the number of states, RNN dynamics, and so on. For these reasons we will not attempt to capture this aspect of rule extraction performance in great detail and will evaluate the algorithmic complexity of the different methods by measuring overall extraction time only.

5

Experimental Setup

As mentioned in chapter 3 we will explore the practical feasibility of the RNN rule extraction methods by applying them to a RNN that is not specifically trained for this purpose. With these experiments we want to answer the question if the methods can be applied to the case study RNN, and if so, how well the resulting automata can approximate the RNN's behaviour in terms of fildelity. Also we hope to gain insight into how each method's hyper-parameters can be selected and how the methods compare in terms of the quality aspects mentioned in the previous chapter. Furthermore, we want to asses how well the automata extracted by each method can help to gain insight into the RNN's operation. This aspect is, as mentioned before, more subjective, but ultimately determines the practical usefulness of each method. We will asses this aspect of the extraction methods by looking at the size of the extracted automata and by giving a subjective judgment on their interpretability. Again, this evaluation is highly subjective, but our research goal is to *explore* the practical use of the RNN rule extraction methods, not necessarily to produce only 'hard' figures, so we feel this is justified.

To answer our research questions we have broken down the research in three main sections, as described in chapter 3. The first part deals with the practical issue of how the continuous input domain of our case study RNN can be discretized and is described in chapter 6. The second part, covered in chapter 7, looks into how the WA method can be applied to a binary classification model, which is not its intended use case. Finally, the third part of our research, applying the methods to our case study RNN, is handled in chapter 8.

All these experiments use the same general experimentation setup, RNN models and datasets. For this reason they are covered in the following sections and not repeated in the subsequent chapters.

5.1. General Setup

For our experiments we used a cloud based computing environment as we suspected that the extraction techniques and RNN implementations might have a high demand on memory and computing resources. After evaluating several options we decided to use Google Colaboratory, which is a free Jupyter notebook environment that requires no setup and

runs entirely in the cloud [Goo20]. A Jupyter Notebook is an open-source web application that allows to create and share documents that contain live code, equations, visualizations and narrative text [jup20]. A Google Colaboratory Notebook gives access to powerful computing resources and combines code blocks with narrative text making it a great option for our research. A Google Colaboratory Notebook has also been used by Weiss et al. [WGY18b] to allow easy access to their DFA rule extraction method [WGY18a].

5.2. CASE STUDY RNN AND DATASET

During the course of our research we will be evaluating the pedagogical rule extraction techniques on a recurrent neural network trained to detect botnet related network traffic. This network was the result of a master's thesis by Tho Poon [Poo18]. In his master's thesis Poon [Poo18] investigated the use of the LSTM and bidirectional LSTM (BLSTM) architectures for the task of botnet detection based on sequences of low-level network traffic features. A selection of 12 fields from the IP, TCP and UDP headers of the individual network packets where used as input features. The studied RNN networks had an input layer with 12 input neurons, to accommodate the input features. The output layer consisted of a single neuron with a logistic sigmoid activation function. The sigmoid activation function gives a real valued output between 0 and 1. A classification threshold of 0.5 was applied to perform the binary classification. Poon experimented with networks with one and two hidden layers of varying sizes and evaluated their performance on different datasets with botnet and non-botnet traffic. The overall best performing network was a RNN with one BLSTM layer consisting of 384 neurons. It is this network that we intended to use as our case study RNN throughout our research.

It is worth noticing that the best performing network still only had an average accuracy of around 80.5% over the evaluated datasets. Also the network had quite a high false positive rate of around 27.3%. This means, as Poon concludes, that the used RNN architecture and sequences of low-level network features may not be well suited to perform reliable botnet detection. Despite these limitations it is still interesting to use this network in our research as we are not looking to accurately replicate or generate the original domain, but rather want to sufficiently approximate the complex behavior of the trained RNN with a representation that is more humanly comprehensible. As Poon himself notes in his conclusions "It is difficult to determine what really happens inside the RNN". Using rule extraction may give us these insights and provide us with a way to explain the behavior of the network and may even shed light on why its performance is subpar.

5.2.1. CASE STUDY RNN RE-IMPLEMENTATION

We originally intended to use the RNN implementation of Poon as-is, however, it proved necessary to re-implement the network in a different technology to make it compatible with our experimentation environment and the rule extraction methods. We used a Python machine learning library called Keras [1] to re-implement the case study RNN. The details on this re-implementation are described in appendix A. We will only repeat the final network architecture and summarized training results here.

[1] https://keras.io/

The architecture of our final Keras based case study RNN is shown in table 5.1.

Layer	Type	Size	Description
1	Input	12	Input layer with 1 node per feature
2	Masking		Masks padded input so network is only trained on real data
3	LSTM	12	LSTM RNN layer with 12 nodes
4	Batch normalization		Aids training convergence and allows larger learning rates to be employed [IS15]
5	Dense (Sigmoid activation)	1	Outputs a continuous value between 0 and 1 representing the positive class probability.

Table 5.1: Case study RNN network architecture

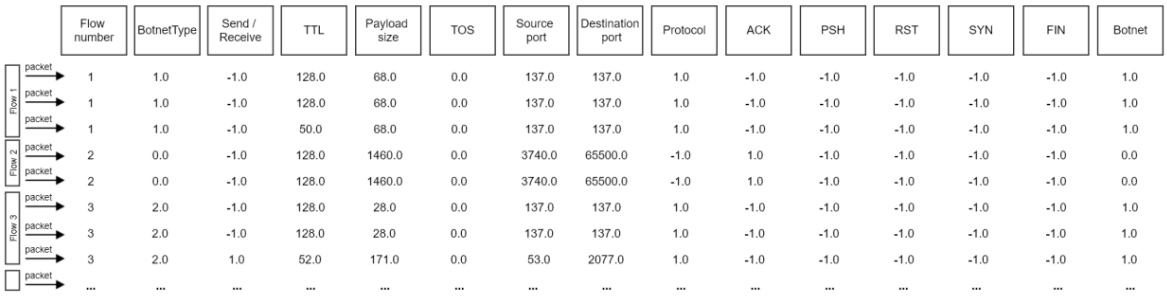
Using the converted datasets, described in the next section, we successfully trained this RNN to 99% accuracy on the training set and 94% on the test set. Surprisingly our RNN far exceeds the performance reported by Poon [Poo18] for his version of the botnet detection RNN. His implementation had a accuracy of just 80.5% and a F1-score of 75.2%. The reason for this quite substantial difference is not entirely clear, but even after reevaluating our RNN implementation and training code and repeating the RNN training several times, we came to the same performance figures for our implementation of the botnet detection RNN. A more detailed analysis of the performance difference can be found in appendix A.

5.3. DATASET

Some of the rule extraction techniques require access to a labeled dataset during the extraction process. For this purpose we will use the original datasets used by Poon [Poo18] to train the studied botnet detection RNN. For our experiments we will use one of the datasets used by Poon [Poo18] in his research on botnet detection. We selected the ISOT dataset as this was the dataset on which Poon obtained the highest accuracy and Poon places some remarks against the validity of the other ISCX dataset.

The ISOT dataset contains flows of botnet and non-botnet network traffic data. In his research Poon uses 12 different fields from the IP, TCP and UDP headers as input features and interprets each packet as a time step in the complete flow sequence. The original ISOT dataset has been split by Poon into a training, validation and test set consisting of labeled sequences of vectors formed by these 12 low-level network traffic features. Figure 5.1 shows the structure of the flows used by Poon. The first two columns are not used as input features and are omited from the final netCDF files Poon created.

Because we had to re-implement the case study RNN we could not use the original netCDF datasets of Poon directly and converted them into a compatible format. All the data was padded to the length of the longest sequence as the Keras training routine expects all the training samples to be of equal length. The datasets contained some very long sequences that would require a large amount of padded data. We set the maximum sequence length to 100 and filtered out all samples that contained longer sequences. This resulted in a loss of samples of less than 1% on the training and validation sets and less than 2% on the training set, which we deemed acceptable. More details on this dataset conversion can be found in



	Flow number	BotnetType	Send / Receive	TTL	Payload size	TOS	Source port	Destination port	Protocol	ACK	PSH	RST	SYN	FIN	Botnet
Flow 1 packet	1	1.0	-1.0	128.0	68.0	0.0	137.0	137.0	1.0	-1.0	-1.0	-1.0	-1.0	-1.0	1.0
Flow 1 packet	1	1.0	-1.0	128.0	68.0	0.0	137.0	137.0	1.0	-1.0	-1.0	-1.0	-1.0	-1.0	1.0
Flow 1 packet	1	1.0	-1.0	50.0	68.0	0.0	137.0	137.0	1.0	-1.0	-1.0	-1.0	-1.0	-1.0	1.0
Flow 2 packet	2	0.0	-1.0	128.0	1460.0	0.0	3740.0	65500.0	-1.0	1.0	-1.0	-1.0	-1.0	-1.0	0.0
Flow 2 packet	2	0.0	-1.0	128.0	1460.0	0.0	3740.0	65500.0	-1.0	1.0	-1.0	-1.0	-1.0	-1.0	0.0
Flow 3 packet	3	2.0	-1.0	128.0	28.0	0.0	137.0	137.0	1.0	-1.0	-1.0	-1.0	-1.0	-1.0	1.0
Flow 3 packet	3	2.0	-1.0	128.0	28.0	0.0	137.0	137.0	1.0	-1.0	-1.0	-1.0	-1.0	-1.0	1.0
Flow 3 packet	3	2.0	1.0	52.0	171.0	0.0	53.0	2077.0	1.0	-1.0	-1.0	-1.0	-1.0	-1.0	1.0
packet	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...

Figure 5.1: Feature file structure

appendix A.

5.4. TOMITA GRAMMARS

In the rule extraction literature it is quite common to evaluate different rule extraction approaches using a set of seven regular grammars originally suggested by [Tom82] and referred to as the *Tomita grammars*. These grammars have been widely studied [OG96; Wan+18b] and consist of well defined regular grammars with varying levels of complexity. The grammars also have know ground truth DFAs which makes them well suited for rule extraction research. We will use these grammar models in part of our research and will therefore describe them in more detail in this section.

The Tomita grammars all have the alphabet $\Sigma = \{0, 1\}$, and generate an infinite regular language over its Kleene closure Σ^*. Table 5.2 describes the strings accepted by each Tomita grammar. Figure 5.2 shows the minimal DFA associated with each grammar.

G	description
1	1^*
2	$(10)^*$
3	an odd number of consecutive 1's is always followed by an even number of consecutive 0's
4	any string not containing "000" as a substring
5	even number of 0's and even number of 1's
6	the difference between the number of 0's and the number of 1's is a multiple of 3
7	$0^*1^*0^*1^*$

Table 5.2: Description of the seven Tomita grammars

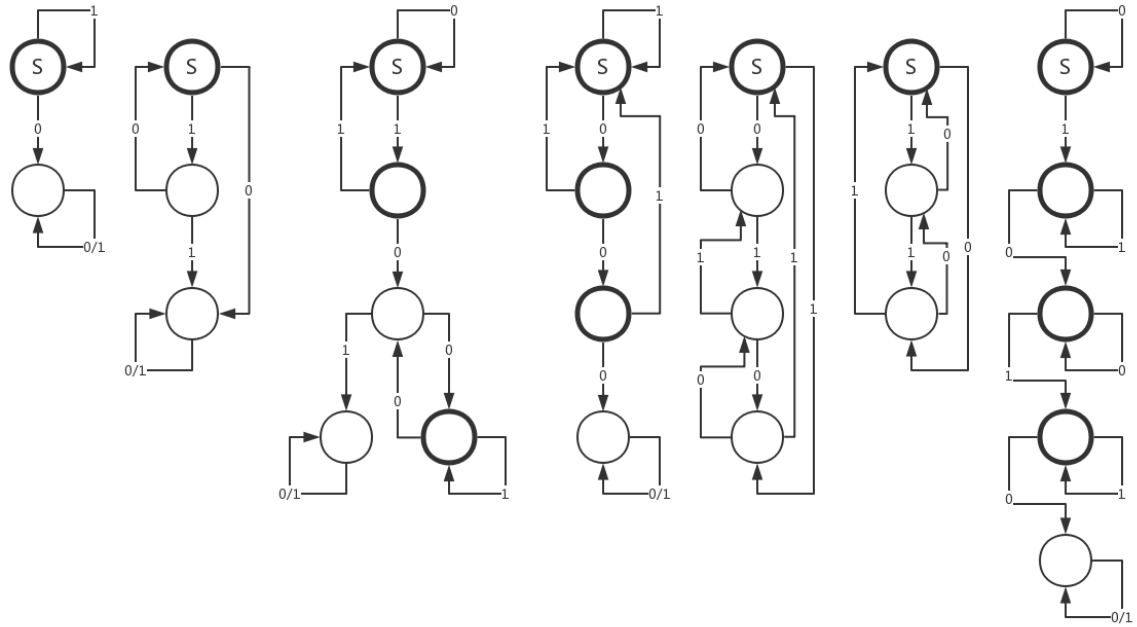


Figure 5.2: The minimal DFA for Tomita grammar 1 to 7, from left to right.

6

INPUT DOMAIN DISCRETIZATION

The RNN rule extraction techniques require the RNN to accept symbolic data as input. Clearly this is not the case for our case study RNN, and may well not be the case in other practical applications. Also the rule extraction techniques need access to a 'symbolized' dataset to some extend. Either to generate initial examples for the negative and positive classes, as is the case for the DFA method, or in the case of the WA method for the generation of the Hankel basis. Furthermore, we also need a 'symbolized' version of the test set to evaluate the fidelity of the extracted automata. So one of the practical challenges that we need to be overcome in applying these rule extraction techniques to our case study RNN is to transform its input domain into a symbolic alphabet. Additionally, we also need to find a way to then use this alphabet as input to the RNN to perform predictions.

The original input domain of the botnet detection RNN consists of vectors of 12 continuous features. To transform this input into symbols and use these symbols as input to the RNN we have experimented with different discretization approaches. The next sections will describe the highlights of these experiments, more detail on these experiments can be found in appendix B.

6.1. ALPHABET SIZE VERSUS INPUT FIDELITY

As with any discretization problem we need to make a trade-off between the size of the resulting alphabet and the amount of information that is lost in the transformation. The size of the input alphabet is a factor in the comprehensibility of the extracted automata, as mentioned previously. The larger the alphabet the more difficult it may be to interpret the semantics encoded by the extracted automata. As it is not clear upfront what the optimal alphabet size will be, in terms of comprehensibility and RNN performance, we quite arbitrarily took 100 as a starting point as this seemed like a reasonable value.

6.2. MAPPING INPUT DOMAINS

As we want the most widely applicable approach we ideally would like to use the RNN on which we perform the extractions as-is, meaning, without any modifications or retraining.

This means that once we have transformed the original input domain into a symbolic alphabet, we need a way to transform the symbols back into usable features for the RNN do it can perform its predictions. To achieve this objective we have to define a symmetrical function or bijection between the original input domain, consisting of 12 real valued features, and a set of symbols. This allows samples from the original input domain to be mapped into symbols and inversely mapped back into vectors in the original input domain. This process effectively discretizes the original samples using the alphabet size as discretization level.

So given an n dimensional input domain I consisting of real valued vectors and an alphabet Σ consisting of m symbols we are looking for the mappings:

$$f : I \rightarrow \Sigma = f : \mathbb{R}^n \rightarrow \Sigma \tag{6.1}$$

$$g : \Sigma \rightarrow I' \tag{6.2}$$

where $I' = \{x \in I\}$ and $|I'| = m$

The inverse mapping, that is the mapping from symbols to input vectors, is used as an additional input adapter to the RNN. The RNN and the input adapter together ultimately form the model that will be exposed to the rule extraction techniques. Figure 6.1 shows this approach.

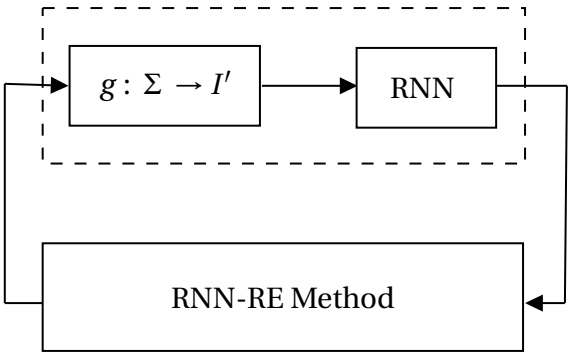


Figure 6.1: Rule Extraction experiment setup

To find such a mapping that would allow us to use the setup of figure 6.1 we conducted several experiments in which we evaluated two different methods to create a symmetrical mapping between the original feature vectors and a set of 100 symbols.

6.2.1. EXPERIMENTS AND RESULTS

In the experiments we used the datasets described in chapter 5. We created 'symbolized' versions of these datasets by passing the samples from the original datasets through the input vector to symbol mapping we were evaluating.

It is difficult to asses the overall affect of the input discretization just by looking at the input data itself, as the RNN has learned a highly complex function between input and output and it is unclear how a certain reduction of input fidelity will affect the classification performance. Therefore, we assessed the affect of the input discretization on the RNN's

classification performance by evaluating the performance of the combined model on the 'symbolized' training set against that of the RNN on the original training set.

The first method we evaluated employed a simple hashing approach with even weight across all input features to discretize the input domain. For the inverse mapping we used the mean values of the individual feature values over all the vectors that where mapped to each symbol. This hashing based input adapter achieved an accuracy of just 72% on the RNN's training set. This is much lower than the 99% of the RNN when applied directly to the training set. A drop in performance was to be expected, but the F1-score on the negative class was also extremely poor.

The second approach employed a clustering algorithm called *k*-means. *K*-means is an unsupervised learning algorithm that can divide input data into a number of distinct clusters. The algorithm will cluster input vectors around the nearest mean values in such a way as to minimize the within cluster variance. This algorithm is used in many scenario's where data needs to be separated into distinct bins or clusters and is widely applied in the literature [Jai10]. For the inverse symbol to feature mapping we used the vector values of the established cluster centres.

For this clustering based approach we fitted the *k*-means model on the original training set and applied it to all datasets. This way our adapter is usable in a setting where we do not know the data upfront, other than the original training data. This is a necessary requirement as the rule extraction approaches can generate input samples during extraction that may represent previously unseen data. Also, if we would fit the *k*-means model on each dataset separately we cannot guarantee consistency in the feature to symbol mapping across the datasets. This could lead to reduced classification performance as well as interpretation issues when we need to map symbols back to input features when we analyze the extracted automata.

We evaluated the performance of this approach on the RNN's training set and found excellent performance. The RNN with *k*-means input adapter achieved an accuracy of 98%. and a F1-score of 95% for the negative class and 99% for the positive class. When we evaluated the *k*-means input adapter on the test set, however, we did see a significant drop in performance. The accuracy was down to 70% and also the F1-scores for the negative and positive classes were significantly lower.

6.2.2. ANALYSIS

The hashing based approach had very poor performance. Clearly the discretization was removing to much relevant information from the input, causing the RNN to misclassify samples. As we felt that an alphabet of 100 symbols was already quite large with respect the comprehensibility of the resulting automata, we did not want to increase the alphabet size. We therefore decided this setup would not be usable for our research.

The *k*-means approach performed better on the training set, however, it had very very poor performance on the test set. The difference in performance on the training and test set can probably be explained by the fact that the *k*-means clustering model was fitted on the training set and than applied to the validation and test sets. If the data is distributed differently between the datasets this may cause a poor fit of the validation and test sets to the cluster centers inferred on the training set. To see if this may be the case we looked into

the sum of the distances of the samples in each dataset to their closest cluster center as inferred on the training set. This measure is known as the k-means score and is in affect the error measure which the k-means algorithm attempts to minimise during fitting. Looking at these k-means scores we did indeed see that the validation and test set have much larger values than the training set. This makes the k-means input adapter unsuitable for our experiments as the rule extraction algorithms could generate samples during extraction that may not yet have been encountered, which may lead to poor translation into the RNN's input domain. This distortion may influence the performance of the extraction process itself and the reliability of the results. For this reason we decided not to proceed with this approach.

In turns out that data discretization is a challenging task, however, it is also an important task that has great impact on all subsequent steps. There are many different discretization algorithms and approaches described in the literature [Liu+02; Jai10] which might perform better, but time did not permit us to further investigate this aspect as it is not the main focus of our research.

6.3. RNN (RE-)TRAINING ON SYMBOLS

As the attempts to create a separate input adapter did not produce acceptable results we re-implemented the case study RNN and changed the input layer to accept the symbols inferred by the k-means approach directly. The idea behind changing the RNN to accept the symbols directly and re-training it on the 'symbolized' training set is that this way the RNN is trained on the discretized data directly which may lead to better classification performance as the network can adjust itself to the different input domain.

Of course we would like an extraction approach where we can use a RNN as-is, but it seems that applying a separate input adapter is not a viable option, at least not for our botnet RNN. We experimented with three different configurations for the input layer of the modified case study RNN to see which setup performed best.

6.3.1. EXPERIMENTS AND RESULTS

The first approach used the symbol indexes directly as a single input feature, but this setup was unsuccessful as the RNN failed to train properly.

The second approach used a one-hot encoding on the symbol indexes, which is a commonly used approach in machine learning when dealing with categorical input features [GB16]. This encoding translates a symbol index i into an array with a length equal to the number of symbols, in which the ith cell has a value of one and the remaining cells are zero. This encoded data transforms the single input feature into, in our case, 100 orthogonal input features. This allows the RNN to adjust all 100 input weights individually to best fit the input data. We successfully trained this RNN setup on the 'symbolized' training set to an accuracy of 99%. We then evaluated the trained RNN on the 'symbolized' test set and found the RNN did not perform well on the test set. Especially the recall on the negative class and the precision on the positive class were quite poor. From these results we concluded that this setup could not be used for our experiments.

The third approach used an embedding layer which turns the symbol indexes into dense

vectors of fixed size. An embedding layer will find similarities between indexes by looking at their context, i.e., the other indexes in a input sequence. The vectors of each embedding get updated during training to express the relationships between indexes. The idea of using this layer is that this additional information on symbol relationship may help the RNN to generalize better and improve its classification performance on unseen data. Although an interesting theory the addition of an embedding layer did not significantly improve the training and classification performance of the RNN compared to the one-hot encoding setup.

6.3.2. ANALYSIS

From these experiments it became apparent that training the RNN on the converted versions of the dataset used by Poon proved difficult. Of course our 'symbolizing' of the data using the k-means approach describe in the previous paragraph will be a contributing factor. However, another cause may be the different class distributions in the datasets. The training and validation set have around 82% positives samples, while the test set has around 45% positive samples. This difference may cause the RNN to generalize poorly on the negative class. Also the fact that the k-means model was fit on the training set and then applied as-is to the other datasets may have caused differences between the datasets which the RNN model failed to generalize on.

To test this hypothesis we redistributed the samples in the datasets to form three new datasets. We combined the original datasets and randomly re-sampled them to form the new training, validation and test set. This way we have a more even distribution of negative samples in the training set and the affect of our k-means discretization approach is 'spread' across all datasets, which may help the RNN to generalize better on the new input domain.

When we trained both versions of the RNN using this new training set and evaluated them on the new test set we did indeed noticed a significant increase in performance. The one-hot encoding version and embedding version of the RNN both trained to an accuracy and F1-scores of around 97% on the test set. We can clearly see that the performance of both approaches significantly improved and that both approaches perform almost identical. We ultimately decided to use the one-hot encoding method for the remainder of our research as it felt simpler, but we could have used either method.

7

APPLYING THE WA METHOD TO A BINARY CLASSIFIER

As mentioned in their paper [AEG18], the WA rule extraction method from Ayache et al. is primarily aimed at extracting WAs from regression type RNN models. Our objective is to evaluate the method's applicability to our case study RNN, which is a binary classifier. This means that we need to explore how we can use the WA rule extraction method on binary classification models and how to interpret the resulting WAs in a binary classification scheme.

To help answer these questions we will evaluate the WA rule extraction method on the seven Tomita grammars first, before we apply it to our case study RNN. The grammar models are small and have known ground truth DFAs which enables us to better evaluate the application of the extraction method itself, without any added uncertainty from the RNN model under study.

In these experiments we aim to answer the following questions:

1. How can the WA extraction method be applied to a binary classifier?
2. Can the WA extraction method achieve successful extraction on the Tomita grammars?
3. How should the extracted WAs be interpreted, i.e., what are their semantics?

7.1. PRACTICAL CONSIDERATIONS

Before we describe the extraction experiments we will first address the practical issues of applying the WA extraction method to a binary classification model.

7.1.1. EXTRACTION ON NON-GENERATIVE MODEL

The code for the WA extraction method, as provided by the researchers, could only be applied to a particular kind of RNN. The code assumed a generative RNN model, meaning it

required a model that can be used to generate strings based on next symbol probabilities. Clearly this does not match our botnet detection RNN or the Tomita grammar models. In their paper the authors mention their method can also be applied to non-generative models by using either the uniform distribution on symbols and a maximum length parameter or by sampling a dataset[AEG18]. We implemented both approaches and added them to the code provided.

7.1.2. SELECTING RNN OUTPUT

Another point that needs to be addressed is the RNN's prediction output values. As mentioned, the WA extraction method is aimed at real valued prediction models. The algorithm builds the Hankel matrices using the output values of the RNN for all combinations of the pre- and suffixes in the basis. For our binary classification model we have two options for these values: We can either take the numeric interpretation of the networks classification output (0 or 1) directly or use the continuous values from the networks output layer before a threshold is applied. As we want to perform the most black-box extraction as possible we will use the first approach in our research.

7.1.3. WA VALUATION THRESHOLD

The WAs extracted by the method of Ayache et al. have continuous weight values for their states and transitions. The automata apply the probability semiring to these values to map input words to real numbers, i.e., to valuate words. As described in section 2.2.2, a semiring is an algebraic structure that defines a set of values and the possible operations on them. The probability semiring uses the usual addition and multiplication operations for the abstract operations. This means applying this semiring gives the product over the summed weights of a run on a word. The automata will thus assign a continuous value to each input word.

As described in 2.3.2 the WA extraction method constructs the weighted automaton using a rank factorization on the Hankel matrices. These matrices are filled with the RNN ouput values for all possible combinations of prefixes and suffixes obtained from the generated basis. This means that the weights in the extracted WA indirectly resemble the different RNN ouputs. As the applied lineair algebraic transformation will have an effect on the original RNN ouput values it is unlikely that the valuations of the automaton will exactly match these values. Therefore, we will have to apply some sort of threshold to their string valuations to mimic the networks binary classification behaviour. What this threshold value should be is not directly clear upfront. However, as the WA is build using the values obtained from the RNN it seems logical to initially use the same threshold as is applied to the RNN output to obtain its binary classification, which was set to 0.5.

7.2. SETUP

For these experiments we performed the WA extraction directly on the Tomita grammar models instead of training a RNN first. By not using RNN models we avoid RNN training, speeding up the experimentation and rule out any influence the RNN models may have. We created a wrapper class that mimics a RNN by exposing a $predict$ method which uses

the Tomita grammar as an oracle. This way the WA extraction method can be used directly on these adapted Tomita grammar models.

We applied the WA extraction algorithm to the adapted Tomita grammar models using the uniform distribution on symbols and a maximum length parameter to generate the Hankel basis. For each extracted WA we additionally performed an ROC analysis to obtain the optimal WA valuation threshold value. We used this threshold value to determine the extracted WAs fidelity against the test dataset using definition 4.8. This way we can guarantee we report the best achievable fidelity for each extraction. We repeated the experiments using the method that samples a dataset to generate the Hankel basis. We used the RNN's training set for this purpose. For each Tomita grammar we performed several WA extractions, using different settings for the WA extraction method's hyper-parameters.

7.2.1. DESCRIPTION OF DATA

To apply the WA extraction method and evaluate the extracted WAs we require training and test datasets. We followed the approach of Weiss et al. [WGY18a] and generated strings of varying length on the Tomita grammars. For each length a large number of strings of that length are are randomly generated and subsampled in such a way as to balance the positive and negative class. This results in a dataset that contains a certain number of random strings for each length specified, as evenly balanced over the positive and negative class as the grammar permits. In their experiments Weiss et al. used string lengths of 0-15, 20, 25, and 30. However, as we also want to compare the different methods to generate the Hankel basis we will use string lengths of 0-20 for the dataset generation and a maximum length of 10, 15 and 20 for the experiments that use the uniform distribution on symbols. This way we can make sure both methods can, in principle, generate strings of equal length. To create a separate training and test set we ran the dataset generation twice.

For the cases where the positive and negative classes are highly imbalanced, as is the case for Tomita grammar 1 and 2 for example, the dataset generation will not provide an evenly split dataset. In these cases additional examples of the minority class can be provided to the dataset generation method to balance out the generated samples. We did not use this feature in our experiments as the WAs could be successfully extracted on the datasets as they where.

7.2.2. AVERAGING RESULTS

During our initial experiments we noticed that the WA extraction results could vary significantly between different runs that use the same settings. Some runs would yield a successful WA, whilst others would yield a WA not even remotely resembling the grammar model. This unpredictable behaviour made it difficult to achieve meaningful results.

To gain more useful results we repeated each experiment 50 times, much like the approach taken by [Wan+18c]. For these 50 runs we report the median performance values and their distributions. We additionally report the percentage of successful extractions, where successful means the extracted WA reached 100% fidelity against the grammar model. In these experiments we kept the training and test set the same for all runs of all experiments on a Tomita grammar. This way we can rule out the influence of any difference

in datasets between experiments.

7.2.3. HYPER-PARAMETERS

The WA extraction method has two hyper-parameters. The first one determines the number of prefixes used to create the Hankel basis and influences that amount of samples drawn from the RNN. It thus influences what data is 'seen' by the algorithm and is used to create the WA. If the basis is too small, not enough representative samples may be seen, if it is too large, the extraction may take an unpractical amount of time and resources. The second hyper-parameter is the rank value used in the Hankel matrix rank factorisation from which the WA is created. It directly influences the number of states of the resulting WA. If the rank value is too low, the WA may not represent the RNN closely enough, if it is too high, the WA will contain a lot of states and the extraction may take longer.

During some initial extraction experiments we have observed that applying a rank value that is too high for the amount of information contained in the RNN will result in a WA with 'unused' states. These states and all their inbound and outbound transitions have zero weight values, effectively making them 'disappear' from the WAs valuations. This indicates that as long as the rank value is high enough the resulting WA will closely match the RNN in terms of fidelity. Normally the appropriate rank value is not necessarily known upfront and will have to be empirically determined. However, as the Tomita grammars have known ground truth DFAs we do have an indication of the appropriate rank values. As weighted automata are in principle non-deterministic, the number of required states for an extracted WA may be smaller than that of an equivalent DFA. We will therefore use the number of states of the DFA for each Tomita grammar as an upper bound for the rank value for its WA extraction.

To find the optimal value for the number of prefixes we started with a relatively low value of 10 and increased it to 25 and finally to 50. As mentioned it is not straightforward to choose the correct basis size upfront, but we felt these values covered the most likely range. Some initial extraction experiments supported this assumption.

7.2.4. CREATING THE HANKEL BASIS

The WA extraction method generates the Hankel basis by sampling words from either a generative RNN model, the uniform distribution on symbols and a maximum length parameter or by sampling a dataset. For each word all its prefixes and suffixes are added to the basis to make it prefix close and the process is repeated until the specified number of prefixes is reached. The words obtained in this manner have a great influence on the extraction results as they determine the information that is seen by the algorithm.

In our case the models are not generative and we have to use either the uniform distribution on symbols or a dataset to generate the Hankel basis. The first methods seems the most widely applicable one, but may, especially in the case of high class imbalance, not yield satisfactory results as it contains no knowledge of the input domain. The second method requires access to, for example, a training set, but is more likely to yield representative words. We will experiment with both options to see which one yields the best result.

7.3. WA TOMITA GRAMMAR EXPERIMENTS AND RESULTS

For word generation from the uniform distribution on symbols we used, as mentioned, a maximum length of 10, 15 and 20. Again choosing the correct value is not trivial, but in this case it is directly related with the maximum length of unique sequences that can occur in the model under study. The Tomita grammars, however, are relatively simple and their maximum sequence length should fall within the selected range.

Some combination of settings may not allow a successful generation of the Hankel basis. For example, a maximum length of 10 may prove to small for a basis size of 50. Where this was the case we indicated the extraction as unsuccessful even if the actual extraction could not take place.

We first report the success rates of the different extraction experiments. Figure 7.1 shows the success rate for each Tomita grammar extraction versus the selected Hankel basis size for the different generation settings. Where an extraction is deemed successful when its average fidelity over the classes is 100%.

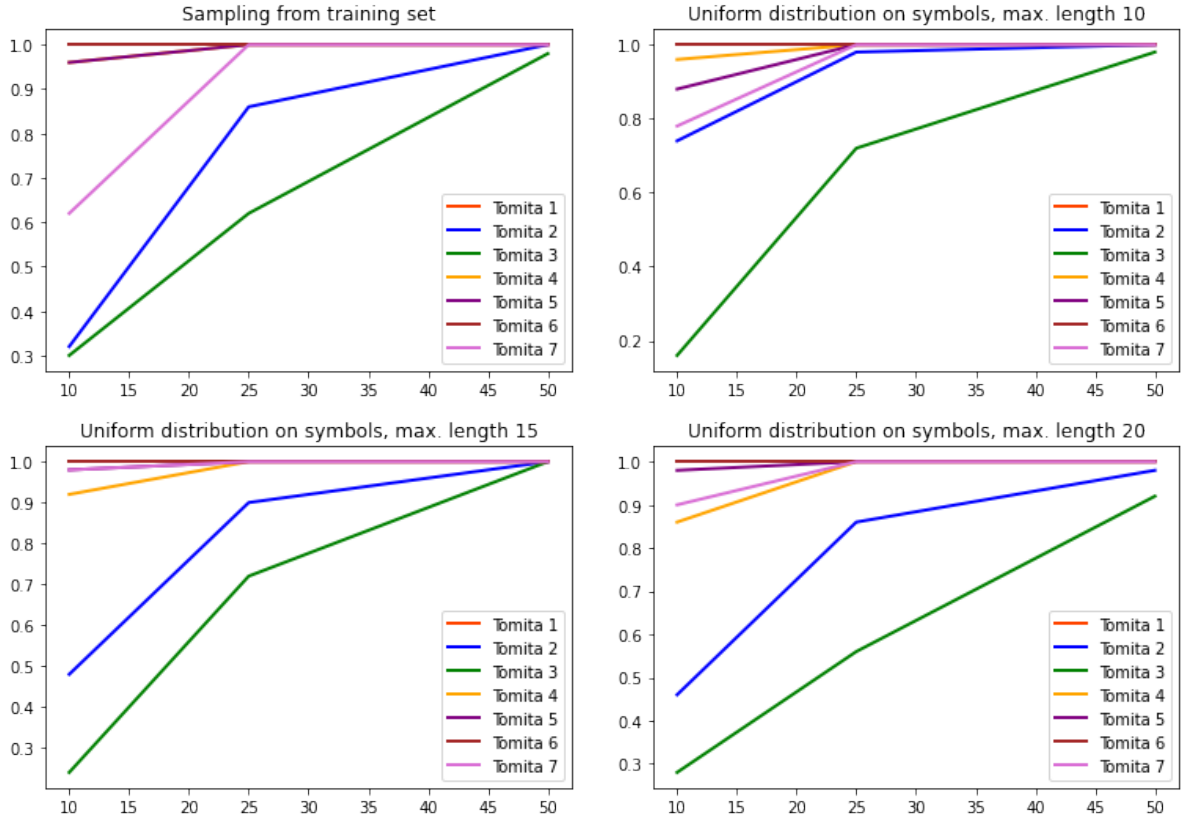
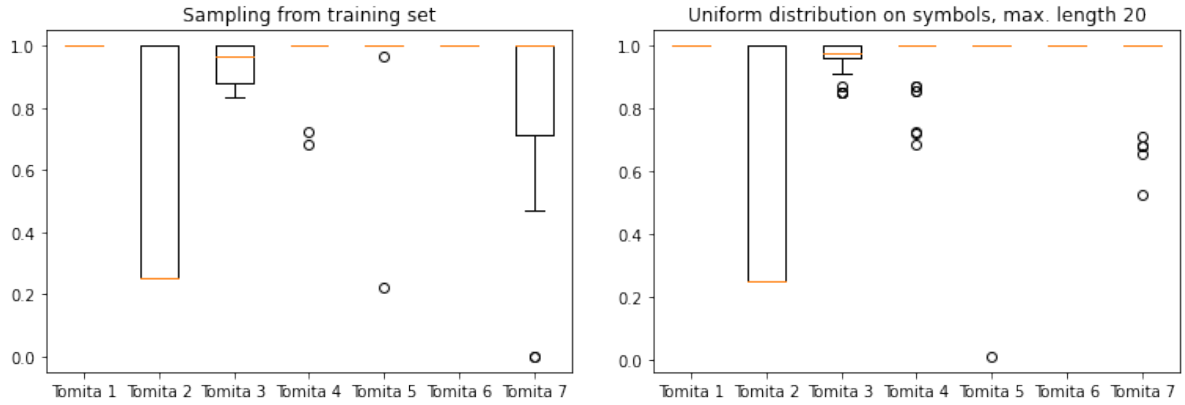


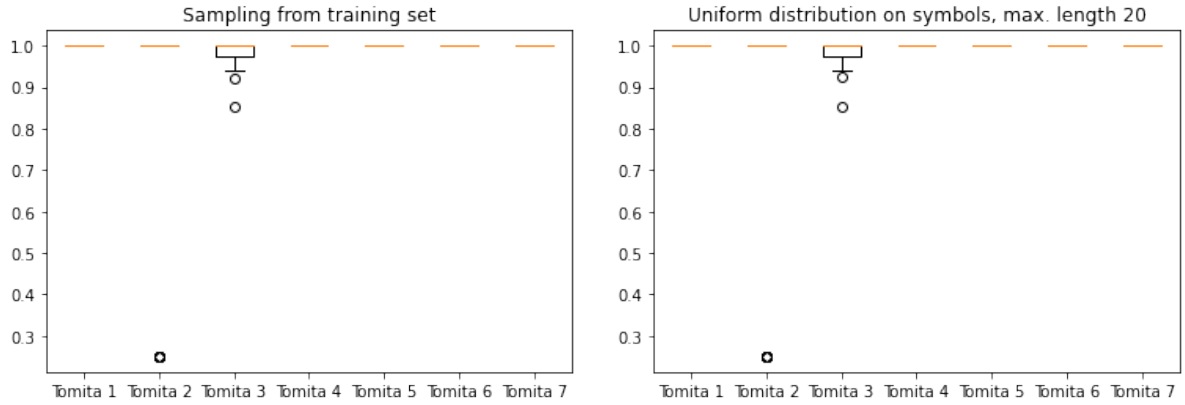
Figure 7.1: Extraction success rate vs. Hankel basis size

As mentioned, the success rate is based on the average fidelity across classes. As a more critical metric we also looked at the minimum fidelity over the classes. This metric will report the fidelity figure of the worst performing class which will make any problems with either the positive or negative class more apparent. For brevity we only report the results for the worst performing extraction settings as they are the most interesting in terms of failure behaviour. We report the results of the extractions for Hankel basis sizes 10 and 25

when using sampling from the training set or the uniform distribution on symbols with a maximum length of 20. Figures 7.2a and 7.2b show the box plots for the minimum class fidelity achieved in the 50 extractions on each Tomita grammar for these settings. These plots show how the individual extractions performed and what the variation of the fidelity figures was over all the successive runs. This way they give an impression of the failure modes of the WA extraction method, i.e., if the method approaches the correct WA in most cases or if it extracts randomly performing WAs between runs.



(a) Minimum class fidelity for Hankel basis size 10



(b) Minimum class fidelity for Hankel basis size 25

We also calculated the difference in fidelity between using the default word valuation threshold of 0.5 or the optimal threshold determined by the ROC analysis. This will tell us if using the default threshold, which is simpler to apply and thus enhances the method's practical applicability, is a viable option. To show the biggest diference we used the minimum class fidelities to determine the fidelity difference. Again for brevity, we only report the result for the settings that gave the biggest difference, being a Hankel basis size of 10 when using sampling from the training set or the uniform distribution on symbols with a maximum length of 10. Figure 7.3 shows the box plot for these differences in the minimum class fidelity. A negative number indicates a worse performance when using the default threshold.

Additionally we report the execution times of the Tomita extractions. We observed quite a large fluctuation in extraction time between runs for some experiments. See figure 7.4 for an example of these fluctuations.

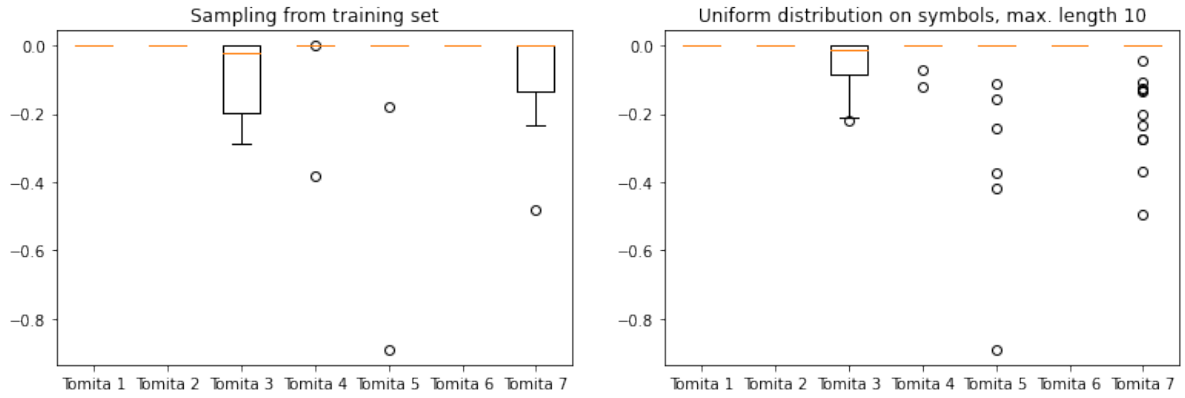


Figure 7.3: Difference in fildelity when using either the default threshold or the optimal threshold

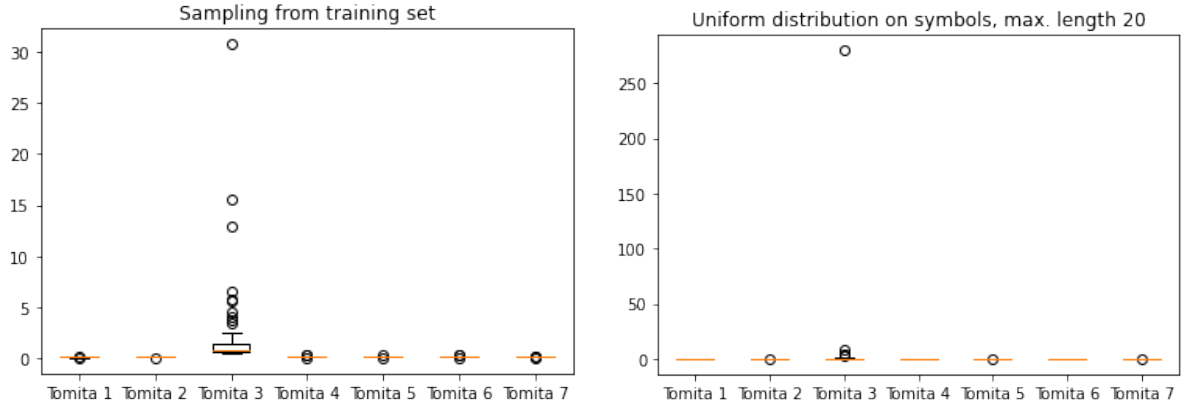


Figure 7.4: Execution time fluctuations (Hankel basis size 50)

Because we want to get an indication of the typical time performance of the WA extraction algorithm we show the norm of the execution times, which shows the most common value and eliminates the effect of the extreme outliers. Figure 7.5 shows the graph for the norm of the execution times. We do have to note that these figures should be used purely indicative as there could be a lot of external factors on the computing platform we used that may influence the execution time. The relative differences in execution time, however, do give a idea of the time performance of the WA extraction algorithm in relation to different settings and model complexities.

Finally we show the extracted WAs for the different Tomita grammars. Figure 7.6 displays representative WAs for each Tomita grammar obtained from the successful extraction runs.

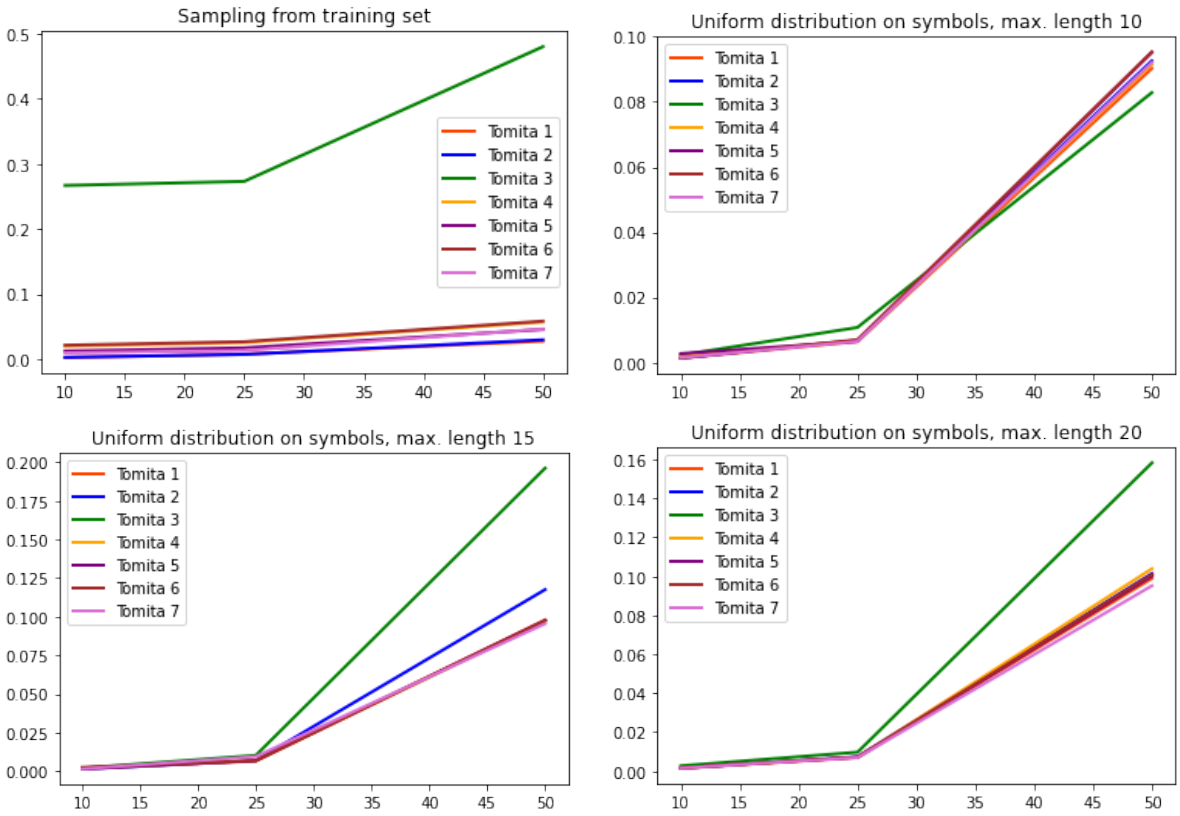


Figure 7.5: Execution time versus Hankel basis size

7.4. INTERPRETING THE EXTRACTED WAS

One of our research questions is to evaluate if the extracted automata provide insight into the RNN's operation. For this to be achieved, clearly, the semantics of the extracted automata must be understood. For the extracted DFAs this is quite straight forward, but for the WAs this is not directly clear. As mentioned, the WAs can be non-deterministic and use the probability semiring to compute their word valuations. These factors makes their visual interpretation quite hard. Even a small WA may be difficult to interpret.

To aid in understanding how an extracted WA comes to a certain output we added a tracing feature to the valuation method. This feature prints a step by step trace of the calculations performed during a word valuation. Figure 7.7 displays an example of such a trace for the valuation of the word 0110 on a (simplified) WA extracted for Tomita grammar 5.

The valuation of a word starts with the vector formed by the initial weights for each state. Each value in this vector is then multiplied by the associated weight value of all outgoing transition from that state for the first symbol. In figure 7.7 this is only the transition from state 0 to state 2 for symbol 0 . The products of all these transitions are then summed per target state, yielding the next state weight vector (see the second line in the table). This process is repeated for all subsequent symbols in the word until the last symbol is reached. The last state weight vector is then multiplied by the vector formed by the final weights for each state to calculate the automaton's resulting word valuation.

For the successful WA extractions we see that some resulting WAs have weight values of

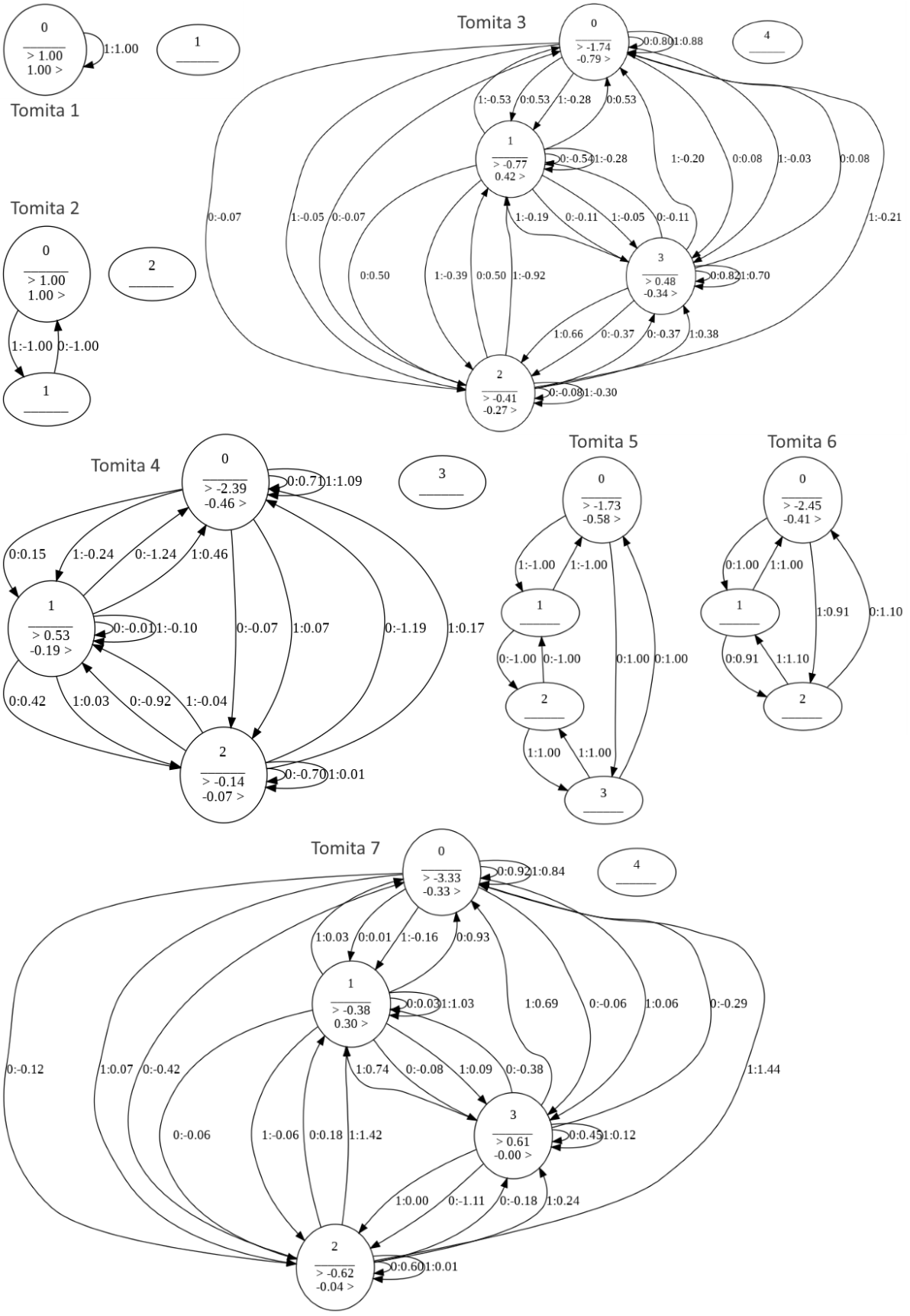
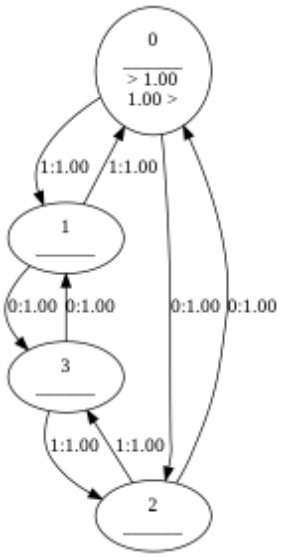


Figure 7.6: Extracted WAs for the Tomita grammars




```
>| [0     , 1     , 2     , 3     ]
   [1.0000, 0.0000, 0.0000, 0.0000]
0  [* -> 0, * -> 1, * -> 2, * -> 3]
   [0.0000, 0.0000, 1.0000, 0.0000]
1  [* -> 0, * -> 1, * -> 2, * -> 3]
   [0.0000, 0.0000, 0.0000, 1.0000]
1  [* -> 0, * -> 1, * -> 2, * -> 3]
   [0.0000, 0.0000, 1.0000, 0.0000]
0  [* -> 0, * -> 1, * -> 2, * -> 3]
   [1.0000, 0.0000, 0.0000, 0.0000]
|> [0     , 1     , 2     , 3     ]
   [1.0000, 0.0000, 0.0000, 0.0000]
=  1.0
```

Figure 7.7: Trace of word valuation on a WA extracted for Tomita 5

exactly 1 on the relevant states and transitions and 0 on all other. Also the WAs are deterministic. See, for example, the WAs for Tomita 1 and 2 in figure 7.6. These WA thus closely follow the reference DFA of the Tomita grammar. In such cases the interpretation of the WAs, once the valuation logic is known, is not too hard.

In other cases, a successful WA extraction may yield a small WA that has, what at first glance looks like, random real values for the relevant weights. However, on closer inspection it turns out that the products of these numbers in the relevant paths still results in a value of (or very close to) 1. See, for example, the WAs for Tomita 5 and 6 in figure 7.6. Again, once the valuation logic is understood these WAs can be interpreted relatively easy.

For the more complicated Tomita grammars, such as Tomita 3, 4 and 7, however, this is a different story. The WAs contain many more transitions, often with small values and are non-deterministic. Also they have multiple entry and exist states, i.e., states that have non-zero initial or final weight values. Even for the relatively simple grammars, such as Tomita 4, their interpretation quickly becomes challenging. A similar situation occurred for the cases where the resulting WA did not achieve 100% fidelity. These WAs often have a larger number of weights with fractional values on their transitions and states and are also often non-deterministic making them more difficult to interpret.

7.5. ANALYSIS

We were able to extract the correct WAs for all grammars using either sampling method method. However, we did experience unpredictable extraction results for the first two grammars. We investigated this phenomenon further and believe that it is caused by the high class imbalance in the first two grammars. Because the Hankel basis is filled with randomly selected samples, a large class imbalance may result in predominately selecting samples of only one class. This will make the data in the Hankel matrices a poor reflection of the actual domain, which would lead to an WA with poor fidelity.

One interesting observation can be made from the success rate figures of 7.1. It seems

that a maximum length of 20 performs worse for a Hankel basis size of 50 than a maximum length of 15 or 10. This feels a bit counter intuitive as a large basis size requires more pre- and suffixes, so larger strings would seem beneficial. We believe this phenomenon is due to the fact that for the larger string sizes, one, or two strings suffice to generate enough pre- and suffixes. However, the combination of those pre- and suffixes may not necessarily properly represent the input space. When a smaller string size is used, more string will be generated before the Hankel basis size is reached. Although shorter, the larger number of these strings allow more variation in the pre- and suffixes increasing the change they sufficiently model the input domain.

Also it seems that for Tomita grammar 3 the minimum Hankel basis size is around 50. All the lower values cause unpredictable extraction behaviour. A similar behaviour can be seen for Tomita grammar 2, although, it seems that when using a maximum length of 10, a Hankel basis size of 25 suffices. The reason for this increased performance on the shortest string length most likely has the same explanation as mentioned before.

The effect, that a larger Hankel basis size improves the success rate, is also present for the other grammars, but less so. For all grammars, except 2 and 3 it seems a Hankel basis size of 25 is sufficient to achieve predictable extraction in all scenario's.

From these figures it seems that using the uniform distribution on symbols with a short string length gives the best extraction performance. However, when the Hankel basis size is large enough for the model under study, using sampling from a dataset also performs well. From this it seems that getting the Hankel basis size right is still the most important aspect of applying the WA extraction method in practice.

We have also seen in these experiments that the WA extraction method not always successfully extracts the right WA, even when using the same settings. This unpredictability between extraction runs has practical implications. As it is difficult to know upfront if the extraction will be successful there is really no other way than to perform repeated extractions and use a cross-folding approach to retrieve the appropriate settings. This means that, although the method seems to be quick, at least on the simple Tomita grammar models, the overall extraction time to achieve a successful WA may be quite long.

In these experiments we have explored how the WA rule extraction method can be applied to a binary classifier. From the results of the experiments we can see that it is indeed possible to use the method on a binary classifier. However, the experiments have also shown that successful extraction depends highly on using the appropriate extraction settings. Especially the size of the Hankel basis has proven to be an important parameter, which was to be expected as it directly determines the information seen by the extraction algorithm. Finally, we have evaluated the interpretability of the extracted WAs. It seems, that at least for simple models, the WAs can convey the information contained in the model. However, we have also seen that interpretation can be challenging for larger models and that human interpretation quickly becomes impractical when the extraction is not 100% successful. The fact that interpretation can be challenging, even for these small models, does not bode well for the application of the WA extraction method on our case study RNN.

8

Case Study Extraction Experiments

As mentioned in chapters 3 and 5 we will explore the practical feasibility of the RNN rule extraction methods by applying them to our case study botnet detection RNN, which was the result of a master's thesis by Tho poon [Poo18].

In the previous chapters we have explored how the case study RNN's input domain can be discretized into a symbolic alphabet and how the WA extraction method can be applied to a binary classifier. We will use these insights when we will try to apply both rule extraction methods to our case study RNN. Additionally the DFA extraction method of Weiss et al. [WGY18a] places some requirements on the RNN it will extract the automata from. So before we can perform the actual extractions we need to address these practical requirements first. Section 8.1.3 describes this in more detail.

With these case study experiments we want to answer the remaining part of our first research question, if the methods can be applied to the case study RNN, and if so, how well the resulting automata can approximate the RNN's behaviour in terms of fildelity. Also we hope to gain insight into how each method's hyper-parameters can be selected and how the methods compare in terms of the quality aspects mentioned in chapter 4. Finally, to fully answer our second research question, we will asses how well the automata extracted by each method can help to gain insight into the RNN's operation. This aspect is, as mentioned before, more subjective, but ultimately determines the practical usefulness of each method. We will asses this aspect of the extraction methods by looking at the overall size of the extracted automata and by giving a subjective judgment on their interpretability. Again, this evaluation is highly subjective, but our research goal is to explore the practical use of the RNN rule extraction methods, not necessarily to produce only hard figures, so we feel this is justified.

8.1. Setup

For these experiments we used the experimental setup described in chapter 5. Also, as with the WA Tomita experiments, we will repeat the extractions several times and report the average values. Ideally we would have repeated the experiments 50 times as we did for the WA Tomita experiments. However, the extractions took considerably longer on the case study

RNN than on the Tomita grammar models, making this many repetitions infeasible. Therefore we had to restrict the number of runs per experiment to 10. This makes the findings we report more susceptible to outliers or other anomalies. We will therefore not only report aggregated values, but also include their variation across the experiments where applicable.

8.1.1. CASE STUDY RNN AND DATASET

As mentioned, we want to apply the rule extraction methods to our case study RNN, which was described in chapter 5. This RNN was trained on an input domain consisting of 12 continuous network traffic features. The rule extraction methods, however, require a RNN that can accept symbolic input, i.e., words over an alphabet. In chapter 6 we explored how the original input domain from our case study RNN could be discretized and if it is possible to add an input adapter to the RNN that can convert this symbolic input back into the 12 continuous features. This way the original RNN can be used on the 'symbolized' input data. This approach did not produce satisfactory results and a new RNN had to be trained that could consume the symbolic data directly. We used this version of the case study RNN in the experiments described in this chapter. The details of this RNN and its performance are described in section 6.3.

8.1.2. DESCRIPTION OF DATA

In chapter 6 we explored how the original input domain from our case study RNN could be discretized. The best performing approach was a k-means based discretization into an alphabet of 100 symbols. We will apply this method to create 'symbolized' datasets from the original datasets used by Poon. We also saw in chapter 6 that the RNN failed to train properly on these converted datasets and redistributed them. We will use such redistributed dataset during the extraction experiments. The final training set we used contained 33086 samples, of which 64.67% positive. The test set contained 18663 samples, of which 64.45% positive. The datasets are thus quite well balanced across the positive and negative classes.

8.1.3. DFA EXTRACTION METHOD API

The DFA extraction method from Weiss et al. [WGY18a] places some requirements on the RNN it extracts the automaton from. The method needs access to the RNN's internal state through a specific API. The RNN's state is used to build and refine a finite abstraction of the RNN which is used to answer the equivalence queries during extraction. This makes the DFA method strictly speaking not pedagogical, although the state is not used directly to extract the automaton as in the (de)compositional approaches. This requirement means that before we can apply the DFA extracton method we need to implement the required API on our case study RNN. The API specifies the following three methods as taken from the Colab notebook provided by Weiss et al. [WGY18b]:

1. CLASSIFY_WORD(WORD) returns a True or False classification for a word over the input alphabet

2. GET_FIRST_RSTATE() returns a continuous vector representation of the network's ini-

tial state (an RState, as a list of floats)

3. GET_NEXT_RSTATE(STATE,CHAR) given an RState, returns the next RState the network goes to on input character char

The continuous vector representation of the network's initial state must include all hidden values of the network. For a LSTM network this would be the concatenation of the hidden and cell state vectors of each of its layers. The LSTM cell state or c state represents the LSTM cell's memory, i.e., its representation of past input. The hidden state of h state is the LSTM cell's output, i.e., the LSTM cell's 'reaction' on its current input and cell state. The LSTM cell state is controlled by hidden state and cell input. The LSTM cell can 'forget' or 'remember' new information based on previous output (hidden state) and current input.

We implemented this API on our Keras based version of the case study RNN. To asses our API implementation we repeated the extraction experiments on the seven Tomita grammars as performed by Weiss et al. in their paper [WGY18a]. The Tomita grammars have an alphabet that consists of only two symbols, 0 and 1, so we had to adjust our RNN's input layer to this alphabet size. We then trained this version of our RNN to 100% accuracy on data generated from each of the Tomita grammars. Using the extraction algorithm provided by Weiss et al. we where able to successfully extract the correct DFA's from each of these trained RNN's, confirming our API implementation is working correctly.

8.2. CROSS-VALIDATION ON DIFFERENT RNN IMPLEMENTATION

Our objective is to evaluate the practical applicability of pedagogical rule extraction methods. One important aspect of pedagogical rule extraction approaches is their potential to be widely applicable as they should place no (or little) requirements on the RNN's architecture or internal workings. This aspect of rule extraction methods is called translucency, as described in 4.3. To evaluate this quality aspect and determine if the extraction methods somehow depend on the provided RNN implementation, we repeated the experiments with a different RNN implementation.

For these additional experiments we used the RNN implementation that Weiss et al. provided in their experimentation code for the DFA rule extraction method [WGY18b]. This RNN is written in Dynet [1], a popular machine learning toolkit for Python, and has a similar architecture than our Keras based RNN. However, instead of one-hot encoding it uses word embeddings on the input layer . The rest of the network consists of one ore more recurrent layers, followed by an output layer consisting of two neurons, modelling the class probabilities. The final binary predictions are calculated by taking the probability of the positive class and applying a threshold value of 0.5. For the experiments we used a version of this RNN with a single LSTM layer with 12 neurons, the same as for the Keras based RNN. The word embedding dimensionality was left to its default value of 3 [2]. The new RNN was trained using the training code also provided by Weiss et al. [3]

[1] http://dynet.io/

[2] This follows the rule of thumb to use the 4th root of the number of input categories for the embedding dimensionality

[3] More details on the RNN implementation and training algorithm used by Weiss et al. can be found in https://github.com/tech-srl/lstar_extraction

We of course used the same datasets as we used for our Keras based version, however, due to how the training algorithm of Weiss et al. expects the training samples, the number of actual samples used will be less. The training algorithm takes a dictionary of words and class labels. This effectively means that all duplicate words are taken out of the training data. As the original data was disctretized using the k-means method into an alphabet of 100 symbols, it is to be expected that there will be samples that contain duplicate sequences (i.e. words). For the Keras based RNN these duplicates are not removed from the datasets, but, due to the chosen implementation, they will be when we train the Weiss et al. based RNN.

We trained this Dynet based RNN to 100% accuracy and it reached an accuracy of 96% on the test set, which is identical to the performance of our Keras based version. A full classification report for this RNN on the test set is given in table 8.1.

	precision	**recall**	**f1-score**	**support**
0	0.97	0.97	0.97	1112
1	0.95	0.95	0.95	635
accuracy			0.96	1747
macro avg	0.96	0.96	0.96	1747
weighted avg	0.96	0.96	0.96	1747

Table 8.1: Classification report on test set for alternative case study RNN implementation

We noticed considerably better training convergence for this RNN than with the Keras based version of the case study RNN. This may be related to the previous point. If all duplicate samples are removed from the training data the network will not have to deal with potentially conflicting samples, i.e., samples that have different class labels for the same word.

8.3. HYPER-PARAMETERS

Both rule extraction methods have a few parameters that affect their extraction process. We will experiment with different settings for these parameters to evaluate their effect and to find the appropriate settings for our case study RNN. To get a feel for the appropriate and feasible ranges of these hyper-parameters we performed some initial extraction experiments, which will be discussed in the next sections.

8.3.1. WA EXTRACTION METHOD

We found that even with relatively small Hankel basis sizes the WA rule extraction method already obtained quite reasonable fidelity figures. For a Hankel basis size of 10 there were already extractions that could reach a fidelity better than 75%. Therefore we chose Hankel basis sizes of 10, 15, and 20 for our experiments. Furthermore, we found that a rank value of 6 already achieved great fidelity, so we chose this as the maximum value for our experiments.

During the initial experiments we experienced out-of-memory exceptions for some extractions. We noticed that especially when we used sampling from the training set to gener-

ate the Hankel basis, we could only use a maximum Hankel basis size of 20 before reaching the maximum available memory (25GB) on some extractions. When we further investigated this issue we found that is was caused by how the Hankel basis was being generated. When we use sampling from the training set there may be some runs in which samples are drawn with very long sequences. The way the WA extraction algorithm generates the Hankel basis causes it to create pre- and suffixes for the whole sequence, even if this means that their number will exceed the set Hankel basis size. This means that when we use sampling from the training we cannot really control the maximum Hankel basis size. In the datasets we use here, the maximum sequence length is 100 symbols. This means that potentially a very large number of pre- and suffixes is generated, resulting in out-of-memory exceptions. Besides leading to out-of-memory exceptions, even on low basis size settings, this also has implications for the interpretation of the results. Because, although we want to evaluate a certain Hankel basis size setting, the actual basis size that is generated may be much larger than reported.

To address this issue we changed the code that generates the Hankel basis. We included a maximum margin to which the requested number of pre- and suffixes may be exceeded. When a word is sampled that will cause this margin to be exceeded it is rejected and another word is sampled. To prevent an endless loop when all subsequent words will exceed this margin, e.g., when we sample from a dataset with only large words, we added a maximum re-sampling criteria. With this adaptation we can better regulate the actual size of the generated Hankel basis. Additionally, we will also record the actually generated number of pre- and suffixes, so we know exactly what Hankel basis size the results are based on.

8.3.2. DFA EXTRACTION METHOD

The DFA method only really has one hyper-parameter, the extraction time limit. The other parameter controls the initial split depth of the first refinement of the abstraction and can be used to prevent the algorithm from prematurely terminating on small automaton. So, as long as this situation does not occur, the latter parameter can be ignored and left to its default value of 10.

During the initial experiments on the Dynet based RNN implementation from Weiss et al. we experienced out-of-memory exceptions on the DFA extractions. We could only run the algorithm using a relatively short time-out setting (< 100 seconds) to prevent it from consuming all available memory.

The DFA extractions on the Keras based version of our case study RNN were much slower than for the Dynet based version. Therefore, we had to apply different time limits than in the experiments on the Dynet based RNN. We tried to use comparable time limits, with respect to the size of the extracted DFA's, as much as possible.

Also on this Keras based version we ultimately experienced out-of-memory exceptions on some runs of the DFA extraction algorithm. We could not reliably run the extraction for time limits beyond 1000 seconds.

As mentioned, we could only obtain a maximum extraction time of 100 seconds on the Dynet based RNN implementation and 1000 seconds on the Keras based RNN before reaching the maximum available memory (25GB). This resulted in DFA's with between 42 and 73 states. As the ultimate goal is to extract automata that can help to explain the RNN's be-

haviour, we felt this number of states would already be a maximum. Therefore, we experimented with extraction times in the range of 1 to 100 seconds for the Dynet based RNN implementation and 5 to 1000 seconds for the Keras based RNN.

8.4. RESULTS

Using the setup described in the previous sections we applied the RNN rule extraction methods to both versions our case study RNN. For brevity we will only report the most interesting findings.

8.4.1. WA EXTRACTION METHOD

Table 8.2 and 8.3 summarize the achieved average fidelities for the Dynet and Keras based RNN versions respectively. They show the minimum, median and maximum average fidelity values over all runs, together with the applied settings for the hyper-parameters. These tables give a quick insight into the overall performance of the WA extraction method and the best settings for the hyper-parameters.

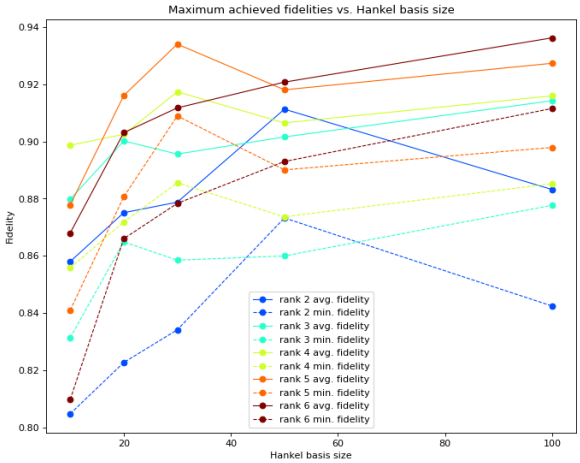
Dynet based RNN						
Rank	Min		Median		Max	
value	avg. fidelity	setting	avg. fidelity	setting	avg. fidelity	setting
2	77%	(100, 15)	84%	(100, 15)	91%	(50, 20)
3	85%	(100, 0)	88%	(100, 0)	91%	(100, 20)
4	83%	(50, 15)	90%	(100, 20)	92%	(30, 0)
5	85%	(100, 20)	90%	(100, 20)	93%	(30, 20)
6	86%	(100, 15)	90%	(100, 20)	94%	(100, 15)

Table 8.2: WA extraction average fidelity per rank value with applied settings. (Hankel basis size, max length), max length 0 means sampling from training set

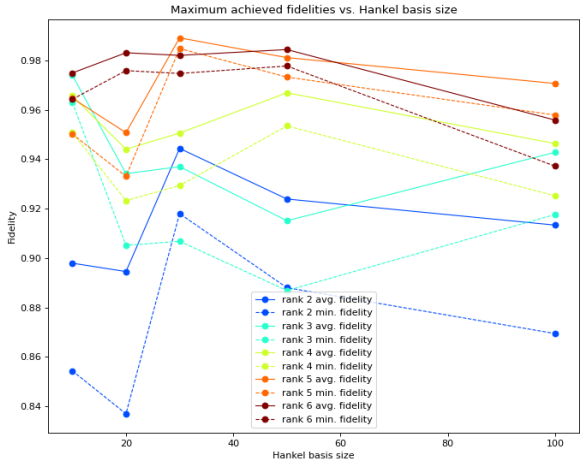
Keras based RNN						
Rank	Min		Median		Max	
value	avg. fidelity	setting	avg. fidelity	setting	avg. fidelity	setting
2	71%	(100, 0)	81%	(50, 0)	94%	(30, 0)
3	77%	(100, 0)	92%	(100, 20)	97%	(10, 0)
4	68%	(100, 10)	89%	(50, 15)	97%	(50, 20)
5	82%	(50, 10)	94%	(30, 0)	99%	(30, 0)
6	80%	(50, 15)	94%	(20, 0)	98%	(50, 0)

Table 8.3: WA extraction average fidelity per rank value with applied settings. (Hankel basis size, max length), max length 0 means sampling from training set

Figures 8.1 to 8.3 give the maximum average fidelities achieved by the WA extraction method for the different settings. The first figure shows the maximum average fidelity achieved for each rank value versus the applied Hankel basis size. The values are calculated by taking the maximum values over all runs for all other settings. This we they represent the maximum fidelity values achieved in the experiments. The next three figures display

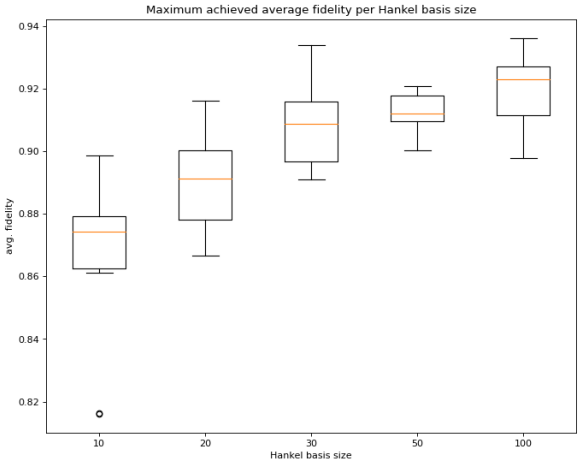


(a) Dynet based RNN

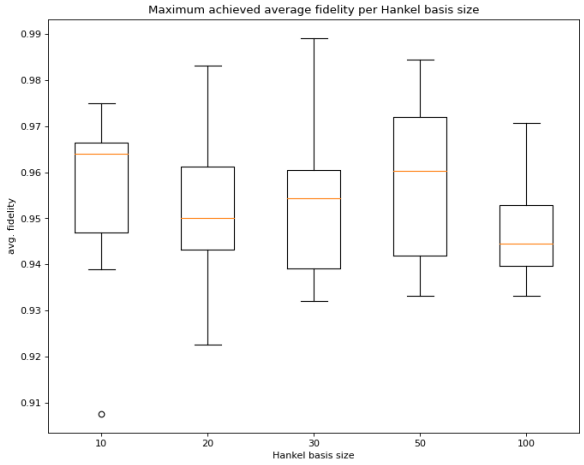


(b) Keras based RNN

Figure 8.1: Maximum achieved average fidelities for WA method.



(a) Dynet based RNN



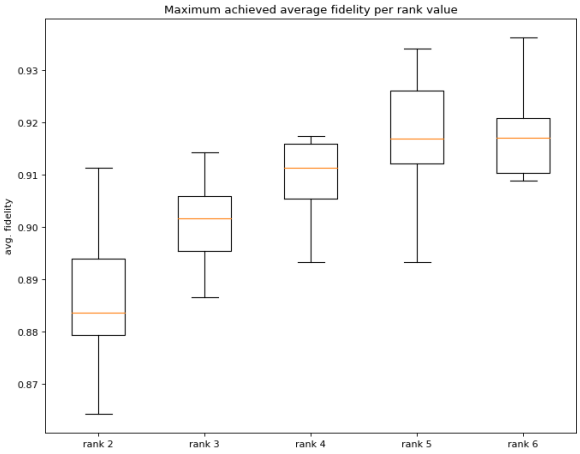
(b) Keras based RNN

Figure 8.2: Maximum achieved average fidelities for WA method per hankel basis size.

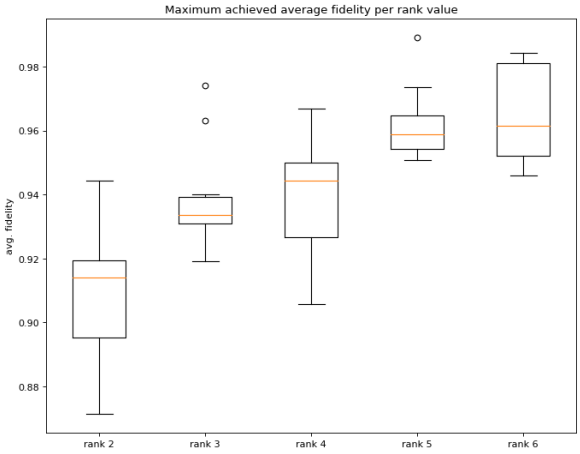
box plots of how the maximum average fidelity varied over all runs, broken down for the different hyper-parameters. The values are calculated by taking the maximum of the runs for all other settings. This we they represent the maximum fidelity values achieved during the runs of the experiments.

Figure 8.5 shows the median of the overall execution time for the WA extraction method per applied Hankel basis size and its variation over the individual runs. Again, the values are calculated by taking the maximum of the runs for all other settings, yielding the maximum values observed during the runs of the experiments. As mentioned in previous chapters, these values should be taken as indicative and not necessarily as representative values for the method in general, as there can be many factors that have influenced the overall execution time.

Finally, in figure 8.6 we show the smallest best performing automata that where extracted in the experiments for both RNN implementations. The images had to be cropped to fit. These automata each have two states and a large amount of transitions between their

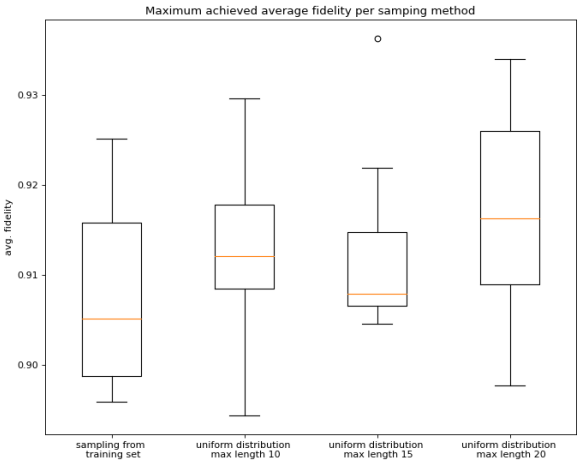


(a) Dynet based RNN

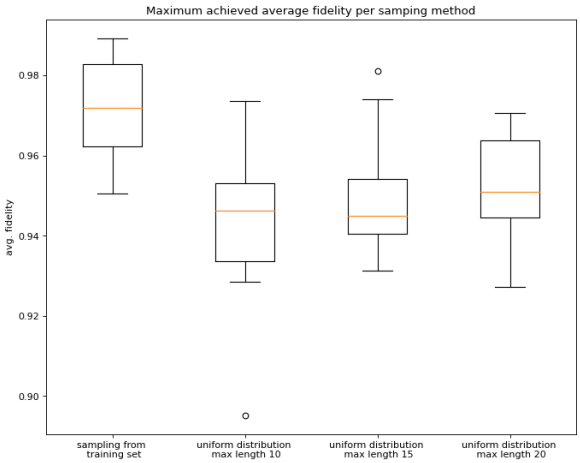


(b) Keras based RNN

Figure 8.3: Maximum achieved average fidelitiesfor WA method per rank value .

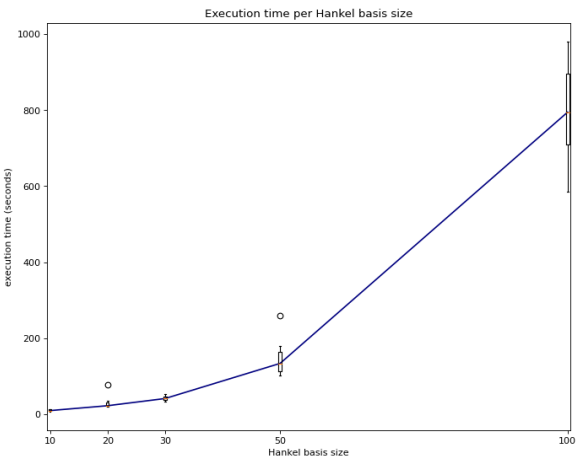


(a) Dynet based RNN

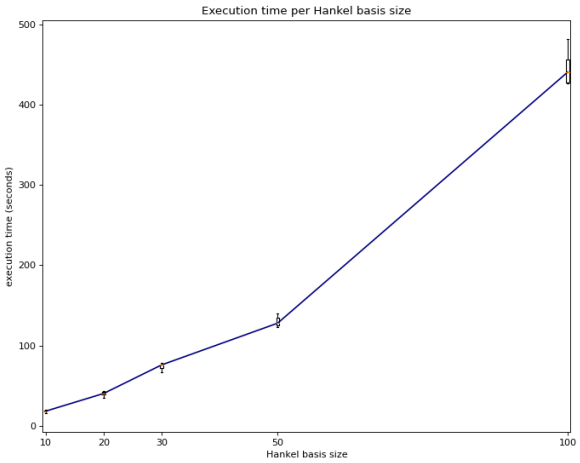


(b) Keras based RNN

Figure 8.4: Maximum achieved average fidelities for WA method per sampling method.

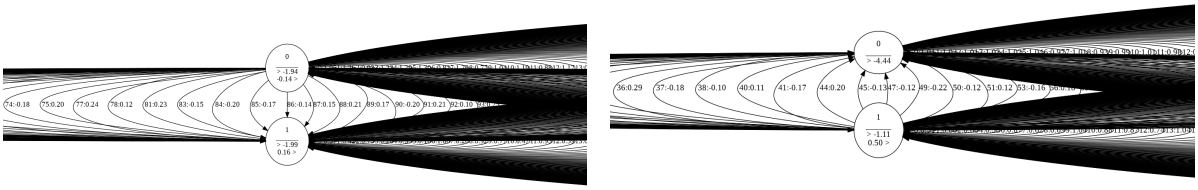


(a) Dynet based RNN



(b) Keras based RNN

Figure 8.5: WA extraction times.



(a) Dynet based RNN

(b) Keras based RNN

Figure 8.6: Best preforming, smallest automata extracted.

states. It appears there is a transition for each symbol in the alphabet between every pair of states, each with a slightly different weight value. The overall best performing automata that where extracted are too large to display here and can be found in appendix C. These automata had 5 to 6 states and an extremely large amount of transitions between the states, again it appears there is a transition for each symbol in the alphabet between every possible pair of states.

8.4.2. DFA EXTRACTION METHOD

The DFA extraction method has less hyper-parameters and no data sampling options. There is really only one parameter that effects the extraction process and influences the resulting DFA and that is the extraction time limit. This means that there are only two dimensions left for which we can report the results, the applied time limits and the runs per experiment. For this reason we report the main results of the experiments in the form of table 8.4 and 8.5.

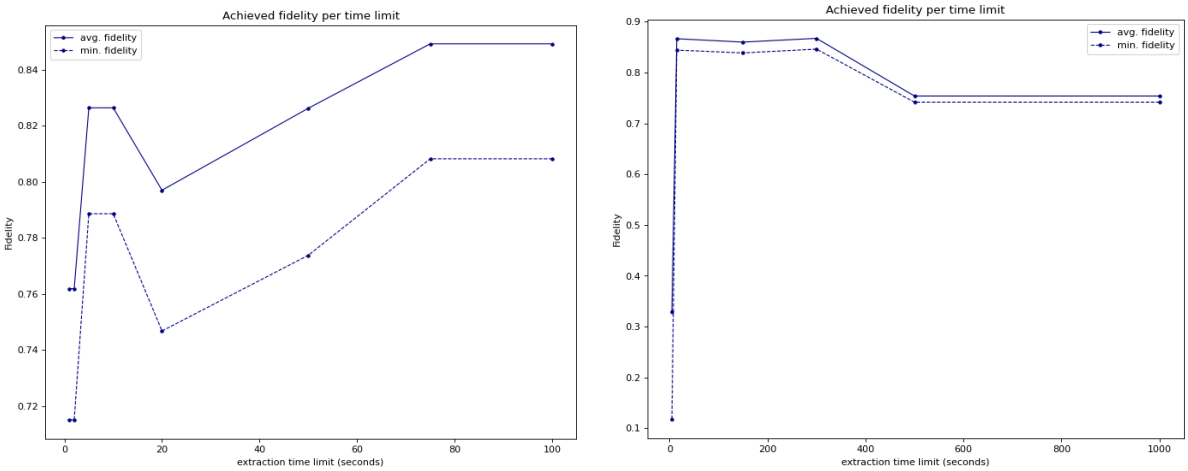
Dynet based RNN						
Time limit	Min		Median		Max	
	avg. fidelity	DFA size	avg. fidelity	DFA size	avg. fidelity	DFA size
1	77%	3	77%	3	77%	3
2	77%	3	77%	3	77%	3
5	84%	10	84%	10	84%	10
10	84%	10	84%	10	84%	10
20	81%	18	81%	18	81%	18
50	84%	36	84%	36	84%	36
75	86%	42	86%	42	86%	42
100	86%	42	86%	42	86%	42

Table 8.4: Achieved average fidelity and extracted DFA size per extraction time limit

Keras based RNN						
Time	Min		Median		Max	
limit	avg. fidelity	DFA size	avg. fidelity	DFA size	avg. fidelity	DFA size
5	27%	2	27%	2	27%	2
15	87%	4	87%	4	87%	4
150	87%	16	87%	16	87%	16
300	87%	39	87%	39	87%	39
500	76%	73	76%	73	76%	73
1000	76%	73	76%	73	76%	73

Table 8.5: Achieved average fidelity and extracted DFA size per extraction time limit

Additionally, we graphically show the achieved average fidelities and DFA sizes for the different time limits in figure 8.7 and 8.8. Figure 8.9 shows the median of the overall execution time for the DFA extraction method per time limit and its variation over the individual runs.

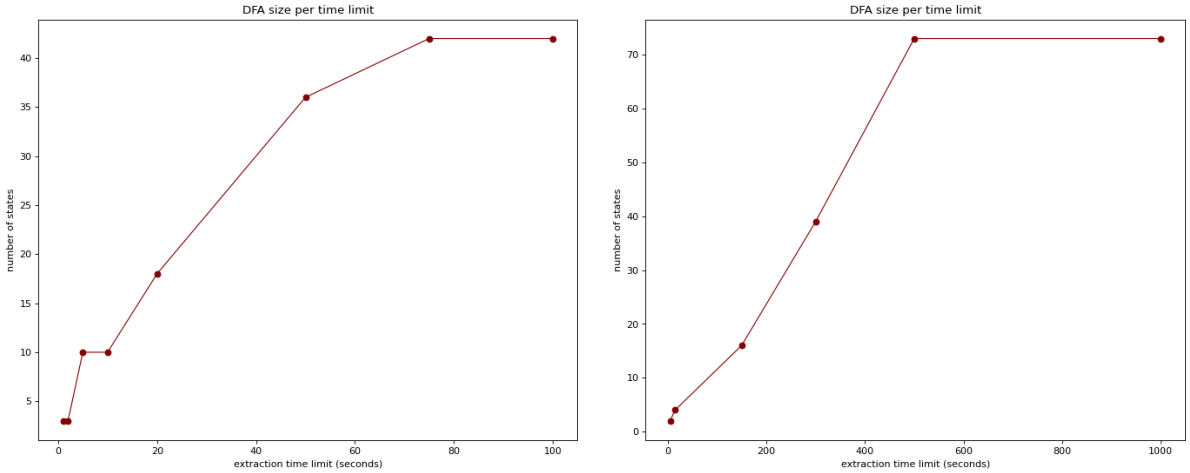


(a) Dynet based RNN

(b) Keras based RNN

Figure 8.7: Achieved average fidelities for DFA method per time limit.

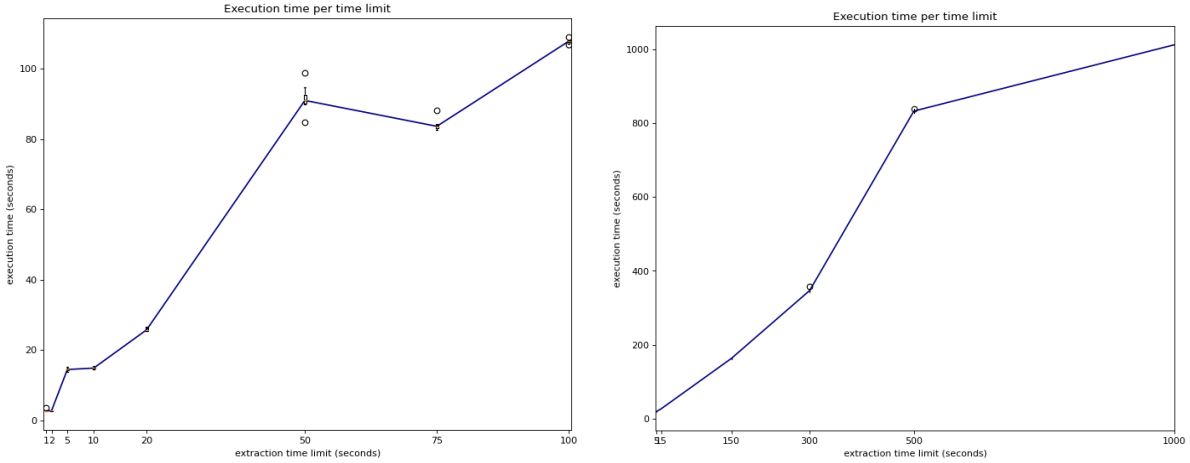
Finally, we show examples of the extracted automata for both methods. Figure 8.10 shows the smallest, best performing automaton that where extracted in the experiments for both RNN implementations. Again, the overall best performing automata that where extracted are too large to display here and can be found in appendix C. These automata had between 42 to 73 states and an extremely large amount of transitions between the states.



(a) Dynet based RNN

(b) Keras based RNN

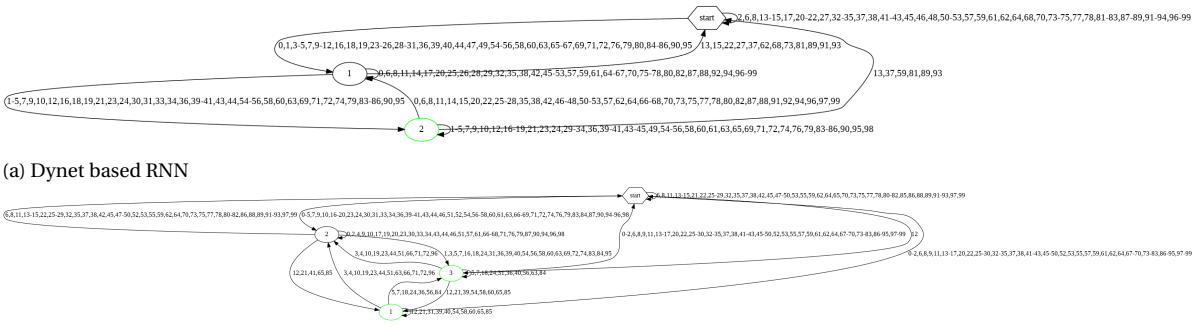
Figure 8.8: Size of automaton extracted by DFA method per time limit.



(a) Dynet based RNN

(b) Keras based RNN

Figure 8.9: DFA extractions times.



(a) Dynet based RNN

(b) Keras based RNN

Figure 8.10: Best preforming, smallest automata extracted.

8.5. ANALYSIS

In general it seems that both methods can successfully extract automata from the RNN implementations. The WA extraction method achieved an impressive maximum average fidelity of 93% on the Dynet based RNN and 99% on the Keras based RNN. The DFA extraction method achieved a maximum average fidelity of 86% on the Dynet based RNN and 87% on the Keras based RNN, which is still quite good. These fidelity figures are surprisingly high and we did not anticipate such high fidelities upfront.

When we look at the extracted automata themselves, however, the situation is not so prefect. The best performing automata, especially the weighted automata, are very complex. They contain a (very) large number of transitions and states, which makes their interpretation nearly impossible. Combined with a large alphabet of 100 symbols and the complex semantics of the weighted automata it is highly unlikely that the extracted automata will provide any real contribution to gaining insight into the RNN's behaviour.

The overall extraction behaviour of the two method is quite different. The DFA extraction method is quite predictable between runs. As we can see from tables 8.4 and 8.5 it will consistently produce the same DFA between runs for a particular extraction time limit setting. The WA extraction method on the other hand, behaves quite erratic between runs. For some runs the method produced a WA that is a good approximation of the RNN, but on other runs it would completely fail to do so. The difference between runs was very large, especially for the lower Hankel basis size settings. Even when we look at figures 8.2, 8.3 and 8.4 which show the best achieved average fidelity we can see there is a large variation between runs.

For the WA extractions on the Dynet based RNN the general trend seems to be that the larger Hankel basis sizes yield a better fidelity. There is a drop in fidelity, however, when using larger Hankel basis sizes for some rank values. For the extractions on the Keras based RNN we see a different trend. Here the larger Hankel basis sizes lead to poorer fidelity values. Especially there is a drop in fidelity for all rank values for a Hankel basis size of 100. This drop in performance for larger Hankel basis sizes may be caused by how the WA extraction method builds the Hankel basis and how the RNNs were trained. The Hankel basis is generated by drawing words from either the uniform distribution on symbols with a maximum lengths or from the training set. The WA algorithm will add all the pre- and suffixes of these words to the list of unique pre- and suffixes. This process will continue until either list's length exceeds the selected Hankel basis size. It then generates words from all possible combinations of the prefixes, single letters from the alphabet and the suffixes. As the generated words are combinations of pre- and suffixes they can be (much) larger than the words from the training set or the maximum length used for sampling from the uniform distribution on symbols. The drop in performance for larger Hankel basis sizes may be partially explained by the fact that the RNN was trained on sequences with a maximum length of 100 symbols. As the WA method can generate longer words for large Hankel basis sizes the RNN may have misclassified these longer sequences which may negatively impact the extraction process.

The WA extraction time on the Dynet based RNN was almost twice as long as for the Keras based RNN. This most likely has to do with the fact that the Dynet based RNN performed single predictions because of how it was implements, whereas the Keras based ver-

sion could use batch prediction. For the DFA extraction time this situation was reversed and DFA extraction on the Keras based RNN took nearly ten times as long as on the Dynet based RNN. This may be explained by the fact that the DFA extraction method preforms single word queries on the RNN and does not group words into batches. This means the RNN have to perform a large number of single word predictions, for which we have found the Keras based implementation took much longer than the Dynet based version. A more detailed analysis on the prediction performance of the RNN's can be found in appendix C.

When we look in more detail at each method individually we see other differences that impact their practical application. The WA extraction method has quite a number of hyper-parameters to get right, especially when applying it on a non-generative model. The appropriate values for these parameters are quite hard to estimate upfront. Also the parameters interact with each other making this even harder, for example, the used sampling method for the Hankel basis generation impacts the Hankel basis size that is actually generated by the algorithm. The DFA method really only has one hyper-parameter, the extraction time limit. This makes it easy to apply in practice. However, we were unable to use large timeout settings as this would lead to out-of-memory exceptions on our environment. One downside of the DFA extraction method may be that the size of the extracted automata cannot easily be controlled by the time limit setting. Ideally each method would provide a means to select the desired granularity of the RNN approximation.

Of course our experiments are not necessarily representative of either method's performance in all scenario's, but we do believe they give a good impression of their practical applicability and the challenges that come with applying them in practice.

9

CONCLUSIONS AND RECOMMENDATIONS

Although recently there have been interesting advances in the explainability of recurrent neural networks in the form of RNN rule extraction (RNN-RE) methods, these methods have been mainly evaluated on small RNN or toy examples and have not really been applied to practical networks. Also, their actual contribution to gaining insight into the RNN's behaviour in a practical setting has largely remained an open question. This means that the real practical use of these methods has been fairly unclear.

This thesis has investigated the practical use of two of the most recent pedagogical, or black-box RNN-RE methods, a WA extraction method [AEG18] and a DFA extraction method [WGY18a]. In chapter 3 we have formulated our main research questions and have broken them down in a number of sub questions. To answer these questions we have explored how the methods can be applied to a botnet detection RNN that has not been specifically trained for the purpose of rule extraction. We have looked into the practical implications of their application and if the extracted automata can ultimately provide an explanation for the RNN's dynamics. The detailed answers to our sub questions are summarized below and are followed by an overall conclusion answering our main research questions.

RQ1.1 *What is the best way to evaluate the overall extraction performance and the quality of approximation of the extracted automata?*

We have answered this question in chapter 4 where we covered several quality aspects and formulated our version of a fidelity metric. We defined an average fidelity metric which gives a good overall impression of the quality of approximation of the extracted automata and additionally we described a more strict minimal fidelity metric, which highlights any problem with individual class performance the automaton may have.

RQ1.2 *How can the case study RNN, with its continuous input, be used by the black-box rule extraction techniques that expect symbolic data?*

We have answered this question in chapter 6 where we explored how the original continues input domain of the case study RNN could be best discretized into a manageable alphabet so it could be used by the RNN-RE methods. A k-means based method performed the best and we managed to infer an alphabet of 100 symbols whilst still maintaining good classification performance.

RQ1.3 *Can the WA rule extraction method, aimed at models that compute a real valued function on sequential symbolic data, be used for binary classification?*

In chapter 7 we investigated how the WA extraction method could be applied to a binary classification model for which it was not originally intended. We tested the method on the seven Tomita grammars [Tom82] and used a ROC analysis to determine the appropriate valuation threshold values. We were able to successfully extract WA's for all grammars and all the automata achieved a fidelity of 100% against the grammar models, proving it is possible to apply the WA extraction method to binary classifiers.

RQ1.4 *Can the techniques be successfully applied to the case study RNN, what are the best extraction settings for each technique and how well can each technique approximate the RNN's behavior?*

We have answered this question in chapter 8 where we applied the methods to two version of the case study RNN and evaluated many different settings for each rule extraction method.

For the WA method we have found that using sampling from a dataset provides the most reliable results. For sampling from the uniform distribution on symbols a maximum length that is just long enough for the selected basis size works best. The WA methods becomes more reliable and produces better performing automata for larger basis sizes. For our case study RNN a basis size of 100 performed best. With respect to the rank values we found that even low settings of 3 already produced good results. The best performance was achieved for a rank value of around 5 to 6.

For the DFA method we have found that really only the extraction time limit needs to be set to achieve working extractions. The method will produce larger automata for longer time limits. For the most parts these larger automata will have an improved fidelity, so to achieve the best fidelity a large time limit would be set. Unfortunately, resource constraints limited us in how large we could set this time limit. However, we do not feel this has limited our findings as we found that even within these constraints the method produced fairly large automata that achieved adequate fidelity.

For the WA extraction method the best average fidelity values we found were 94% for the Dynet based version of the case study RNN and 99% for the Keras based version. The best minimum fidelity values were not too far behind with 91% and 98%, respectively.

For the DFA method the best average fidelity values we found were 86% for the Dynet based version of the case study RNN and 87% for the Keras based version. The best minimum fidelity values were 80% and 84%, respectively.

These figures show that both methods can approximate the case study RNN's behaviour quite well, but that, at least in our experiments, the WA extraction method is superior in this respect.

RQ2.1 *How should the numerical semantics of the extracted WAs be interpreted, particularly in a binary classification scheme*

We have answered this question in chapter 7 where we performed WA extractions on the seven Tomita grammars. We investigated the semantics of the extracted weighted automata and analysed how they should be interpreted.

RQ2.2 *How well does each technique help to explain the RNN's operation?*

In chapter 8 we applied the extraction methods to our case study RNN and looked at the smallest, best performing and overall best performing automata. We found that even the smaller automata were far too complex, either in size or in their semantics, to offer any global insight into the RNN's operation.

We can now answer our main research questions:

Can the black-box rule extraction techniques be applied to a practical RNN not specifically trained for this purpose?

and

Do the extracted rules provide insight into the RNN's operation?

We have found that, although some practical aspects had to be addressed, the methods could be successfully applied to the case study RNN and that they obtained good to excellent fidelity figures. The extracted automata, however, are so complicated, either in size or in their semantics, that they do not readily enable human interpretation. Therefore, we feel that the extracted DFAs and WAs do not easily provide a global explanation of the RNN's operation. They might, however, be useful to analyse the RNN's prediction on a specific datum, i.e., provide a local explanation. Overall we conclude that, although the techniques are interesting and can certainly be useful, they are unlikely, in their current form, to provide a complete answer to the RNN explainability problem.

9.1. LIMITATIONS

The results presented in this work demonstrate the practical application of the two most recent black-box RNN-RE methods. This work, explorative in nature and intended to be a practical evaluation of the extraction methods, has several limitations, either inherent or practical in nature.

9.1.1. GENERALITY

We have evaluated the practical feasibility of the rule extraction methods on one particular model, the botnet detection network. Although we have used two different implementations for this network, the results are still based on the application of the extraction methods on one problem domain. This makes it difficult to draw general conclusions from these experiments. However, the trends that we have found and the insights into the interpretability of the extracted automata seem more general.

Also we evaluated the extraction methods on the 7 Tomita grammars, which have been widely used in other RNN-RE studies. This side-by-side comparison of the different methods under similar conditions does offer some general insights into the practical applicability of the two methods.

Furthermore, the main objectives of this research were to *explore* the practical applicability of the rule extraction methods and to gain *insight* into the interpretability of the extracted automata and to asses if they can help to gain insight into the behaviour of the RNN. We feel we have achieved these goals, even if the results themselves may not be fully generalizable.

9.1.2. RESEARCH CODE

Although we have been as thorough as possible, there is always a change that the research code we wrote contains errors that may have influenced our results. The same applies to the code we received from the researchers of the evaluated rule extraction methods. However, we have repeated the extraction experiments on different datasets and cross-validated on different RNN implementations and still achieved similar results. Also the achieved results seem logical and the trends in the data can be explained, so we feel quite confident that this is not the case.

Also, the Keras based RNN implementation and the associated data prepossessing we wrote may quite possibly not be optimal in terms of performance. This may have had an effect on the rule extraction performance of each method, resulting in worse reported extraction times than would otherwise be possible. Although this may have increased extraction times it does not, at least in our opinion, invalidate or compromise our other findings.

9.1.3. CONSTRAINTS ON HYPER-PARAMETERS

We experienced large memory consumption of both RNN implementations when performing repeated single predictions. This seems to be related to the computation graphs of the underlying technology. Since this problem caused out-of-memory exceptions we had to restrict the maximum Hankel basis size for the WA extraction method and the maximum extraction time for the DFA extraction method. These restrictions limit the potential of each method to extract automaton with (very) high fidelity. Although, a limitation, we feel that, given the recorded fidelity scores of the extracted automata, this has not led to a significant degradation in the experiments we performed.

9.1.4. HANKEL BASIS SIZE

In the case study extraction experiments we uncovered an issue with the Hankel basis generation which could lead to a basis that is much larger than the size requested. To address this issue we changed the Hankel basis generation code and recorded the actual number of pre- and postfixes generated in each experiment. For the WA Tomita experiments, however, we did not do this, since at the time these experiments were conducted we did not know about this issue yet. Most likely this was less of an issue for these experiments because we limited the word sizes in the data sets to a maximum length that was similar to the evaluated maximum string lengths for generation from the uniform distribution on symbols. However, there is still a change that the generated Hankel basis was larger than the evaluated setting. Worst case its size can be almost twice as large. For example, when the previous samples grew the number of pre- and postfixes to just under the Hankel basis size setting and the next sample is a word of the maximum length. As mentioned, we did not

record the actually generated number of pre- and postfixes in the WA Tomita experiments, so we do not exactly know how often this situation has occurred during the experiments. This may have impacted the validity of the results for these experiments.

9.2. RESEARCH CONTRIBUTIONS

The main contribution of this research is to provide insight into the practical applicability of the two most recent pedagogical recurrent neural network rule extraction techniques. We have identified and addressed some practical implications of their application, demonstrated how the methods can be used on a case study RNN and assessed how the extracted automata contribute to explaining the RNN's operation. Also we have performed a comparative evaluation of both rule extraction methods under similar circumstances. This, to the best of our knowledge, has not been done before in this manner.

Additionally, we have made further contributions throughout this research, either practical in the form of changes or additions to the code of the extraction methods, or in the form of insights into their application. We briefly summarize these additional contributions here.

1. We have found and corrected several issues with the WA extraction method's code as provided to us by the researchers. We have added implementations for generating the Hankel matrices based on sampling from a dataset or the uniform distribution on symbols using a maximum length parameter. We have informed the researchers and provided our version of the code to them.

2. We have explored if the WA extraction method could be applied to a binary classifier and proven this is possible. With this insight we may have broadened the method's application.

9.3. FUTURE WORK

We sincerely hope that the work presented in this thesis may spark further research on the explanation of recurrent neural networks or the area of explainable machine learning in general. Based on the work presented, we believe the following directions could be interesting for further research.

Firstly, for the generality of this research, additional experimentation could be done on other RNN models from different domains and for a larger range of settings for each method to see if the results found in this thesis apply to other areas as well. Also, the way we discretized the original continuous input domain of the case study RNN can be improved. We ultimately had to re-implement the case study RNN and directly train it on the 'symbolized' input domain, as we were not able to simply use a separate adapter in front of the original RNN. This, of course, severely limits the practical application of this approach. Perhaps a separate ANN could be used as input adapter and trained to learn the mapping between the symbolized' input domain and the original domain so the original RNN can be used on 'symbolized' data.

Secondly, we have seen, especially when the target class distribution is highly imbal-

anced in the input domain, that the WA extraction algorithm performs poorly with respect to extraction success rate. The main reason for this, we found, is the way the Hankel basis is generated. Because the generation relies, for the case of a non-generative model, on random sampling, there is no way to guarantee that the selected samples are representative for the input domain, i.e., capture all the relevant variations. So any improvements in this area would be highly beneficial. Especially methods that would increase the chance of selecting representative samples, i.e, samples that lead to combinations of pre- and postfixes that form relevant data points in the input domain. Alternatively or related, it would be good to have an upfront indication, i.e, some sort of metric, of how well the selected data-points in the Hankel matrices capture the information of the input domain, meaning before the expensive step of querying the RNN is started. This way the basis generation could be retried, which is relatively inexpensive, without having to perform all the time consuming predictions before finding out the resulting WA has poor fidelity.

Thirdly, for the rule extraction methods we have evaluated in this work we believe the interpretation of the extracted automata could be enhanced. To improve the readability of the automata in general and to enhance the link to the original input domain, the symbols in the automata could be mapped back to the input domain values or value ranges they represent. Also visual highlighting or tracing of paths in the automata may help with local explanations. Furthermore, the non-deterministic nature of the weighted automata makes them exceptionally difficult to interpret. Perhaps incorporating techniques such as determinization [BGW00] could contribute to alleviate this.

Finally, however, from what we have seen in this research we do not believe that either weighted automata or deterministic automata will ultimately be the way to provide insight into the complex operation of recurrent neural networks. Therefore the main direction of further research, we believe, should be towards finding other form of automata that can better capture and convey the behaviour of a trained RNN.

10

REFLECTION

We chose to use an online computing platform to conduct our experiments. Although a well considered choice at the time it has been a mixed experience. It did enable us to work easily from different locations, which was a great help whilst balancing between our work obligations and performing this graduation assignment. On the other hand we also had some problems with fluctuating performance and unexpected terminations of execution sessions. Despite these drawbacks we still feel it was a good decision to use a cloud based computing platform.

Initially we did not anticipate that we would have to develop and train RNNs ourselves. However, quite soon in our research it became apparent that this would be required to be able to achieve the research objectives. At the beginning of this assignment we had very limited knowledge of machine learning, let alone recurrent neural networks. Also we had little to no experience with the required tooling or languages such as Python or Keras. We had to cover a lot of topics in a very short time and we are quite proud of the results we achieved.

Overall the process of performing this graduation assignment had its ups and downs. It has been one of the hardest things we ever had to do, but has been very educative and rewarding in the end.

A

CASE STUDY RNN

Originally we did not intend to re-implement or re-train the case study RNN. The main focus of our research is on RNN rule extraction and not on RNN architectures or RNN training. However, quite early in our research it became apparent that we could not use the botnet RNN from Poon [Poo18] directly. Poon used a RNN toolkit called, CURRENNT, which is an open-source implementation of deep recurrent neural networks written in C++. The pedagogical rule extraction methods that we want to evaluate, however, are written in Python.

In principal it is possible to call C++ code from Python. However, we also wanted to allow our research code to be executed on a cloud platform as we suspected we might need computational resources that would exceed those locally available to us. Besides this platform requirement, we also had no real prior experience with either C++ or Python. We felt it would be to much of a project risk to try and bridge both language systems. Also there was a good possibility that we would need to change or adjust the RNN later in our research. As we felt slightly more comfortable in Python, and this also seemed to be the language of choice of the researches and the machine learning community in general, we decided we would not use the original RNN of Poon, but rather re-implement it in Python.

We first looked into reusing the network files of Poon containing the trained RNN weights so we would not have to re-train our RNN. Unfortunately we were not able to use these weight values to initialize our Python RNN implementation as the exact structure of these weights and how they would correlate to those in our RNN was not clear. Rather than spending a lot of time in understanding the intricate details of the different RNN implementations we decided to re-train the Python RNN using the original training set used by Poon.

After some investigation into the different machine learning libraries and toolkits available in Python we decided to use Keras [ker20]. as this seemed like to most user friendly and flexible one. On its website Keras is typified as: *"Keras is a high-level neural networks API, written in Python... It was developed with a focus on enabling fast experimentation. Being able to go from idea to result with the least possible delay is key to doing good research."*.

Using the Keras documentation we constructed a simple RNN model with an input layer for the 12 input features, a hidden LSTM layer with 12 nodes, and a dense output layer with

one output. This RNN model is different from that of Poon's best performing RNN that was a BLSTM network with one hidden layer of 384 neurons. We felt, however, that it would be a better strategy to start with a simpler model and increase its complexity when required. Besides being a good engineering practice, keeping things simple is also the preferred practice in machine learning to avoid overfitting [Haw04]. Overfitting occurs when the model has to many degrees of freedom and in a way overcomplicates the problem, such a model may fail to generalize correctly on the training data resulting in poor classification performance on unseen data.

The original netCDF dataset of Poon could not be used directly to train our Keras RNN so we needed to convert it into a compatible format. Also the Keras training routine expects all the training samples to be of equal length. To achieve this all samples should be padded to the length of the longest sequence in the dataset using a special padding value. The datasets contained some very long sequences that would require a large amount of padded data. We set the maximum sequence length to 100 and filtered out all samples that contained longer sequences. This resulted in a loss of samples of less than 1% on the training and validation sets and less than 2% on the training set, which we deemed acceptable.

Because we padded the input sequences we also need to add a masking layer to our network. A masking layer annotates the input data so the following layers can distinguish between real and masked data. This prevents the network from training on masked data as well.

We have also experimented with two output neurons, one for each class. In this case the binary classification of the model is achieved by applying an argmax function to the output, which gives the index of the output with the highest value, i.e., the predicted class. Although both configurations could ultimately be trained to reach similar performance we did experience slightly better training convergence using the version with separate output neurons per class. The architecture of our final Keras based network is shown in table A.1.

Layer	**Type**	**Size**	**Description**
1	Input	12	Input layer with 1 node per feature
2	Masking		Masks padded input so network is only trained on real data
3	LSTM	12	LSTM RNN layer with 12 nodes
4	Batch normalization		Aids training convergence and allows larger learning rates to be employed [IS15]
5	Dense (Sigmoid activation)	1	Outputs a continuous value between 0 and 1 representing the positive class probability.

Table A.1: Case study RNN network architecture

Using the converted training set we successfully trained this RNN to 99% accuracy on the training set and 94% on the test set. See table A.2 for a full classification report on the test set showing the precision, recall and F1-score for both the negative and positive class, their averages and the overall accuracy.

	precision	recall	f1-score	support
0	1.00	0.89	0.94	10225
1	0.88	1.00	0.94	8438
accuracy			0.94	18663
macro avg	0.94	0.94	0.94	18663
weighted avg	0.94	0.94	0.94	18663

Table A.2: RNN classification report on test set

MASKING

When we added a masking layer we ran into problems with running the model on the GPU in Tensorflow 2.1. It turned out to be a known bug in the Keras/Tensorflow 2.1 implementation that has to do with fully padded sequences. The problem is that some input sequences only contain padding and after masking will produce an empty sequence, which causes the Tensorflow 2.1 engine to report an error when running on the GPU. At the time of writing this problem has not yet been resolved in Keras/Tensorflow 2.1. To work around this problem we had to resort to running the RNN on the CPU only.

RNN PERFORMANCE ANALYSIS

Surprisingly our RNN far exceeds the performance reported by Poon [Poo18] for his version of the botnet detection RNN. His implementation had a accuracy of just 80.05% and a F1-score of 75.10%. The reason for this quite substantial difference is not entirely clear, but even after reevaluating our dataset conversion code, RNN implementation and training code and repeating the RNN training several times we came to the same performance figures for our implementation of the botnet detection RNN.

A possible explanation might be that our RNN is a LSTM with one hidden layer of only 12 neurons where as Poons best performing RNN was a BLSTM network with one hidden layer of 384 neurons. This significantly larger number of hidden neurons might have caused his network to overfit the training data which may lead to lower performance on the test set [Haw04]. Poon does not provide enough detail in his thesis to further substantiate this notion, however, the large validation errors reported in his work might also point in this direction.

Another potential explanation could be the fact that we normalized the input features using a min/max scaler to be between 0.01 and 0.99 so we could use 0.0 as padding and masking value. The original features were normalized to have zero mean and unit variance, but their actual value ranges differed quite a lot between them. Some features were between -1.0 and 1.0 whilst others where between -0.6 and 5.2 see also B.1.3. We did experience a drop in training convergence and classification performance when we omitted the min/max scaling, which further points in this direction.

B

INPUT DOMAIN DISCRETIZATION

The original input domain of the botnet detection RNN consists of vectors of 12 continuous features. To transform these vectors into symbols and use them as input for the RNN we have experimented with different discretization approaches. In this appendix the results of these experiments can be found that were too large or detailed to include in the main content.

B.1. DISCRETIZATION EXPERIMENTS

Our first approach was to define a symmetrical function or bijection between the original input domain, consisting of 12 real valued features, and a set of symbols. Samples from the original input domain can be mapped into symbols and inversely mapped back into vectors in the original input domain. This process effectively discretizes the original samples using the alphabet size as discretization level.

So given an n dimensional input domain I consisting of real valued vectors X and an alphabet Σ consisting of m symbols we are looking for the mappings:

$$f : I \rightarrow \Sigma = f : \mathbb{R}^n \rightarrow \Sigma \tag{B.1}$$

$$g : \Sigma \rightarrow I' \tag{B.2}$$

where $I' = \{x \in I\}$ and $|I'| = m$

The inverse mapping, that is the mapping from symbols to input vectors, is then used as an additional input adapter to the RNN. This way the original trained RNN can be used as-is, which, in our opinion, would lead to the most widely applicable approach. The RNN and the input adapter together form the model that will be exposed to the rule extraction techniques. Figure B.1 shows this approach.

B.1.1. DATASET CONVERSION

In the experiments we conducted, we wanted to use datasets that contain sequences of symbols, i.e, words, and the associated classification labels. This way we could perform

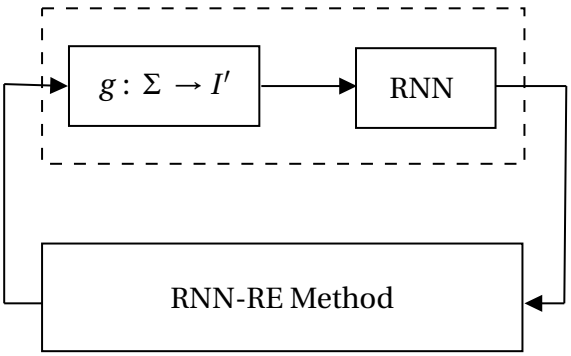


Figure B.1: Rule Extraction experiment setup

extractions on these dataset directly, where possible, and exclude any influence the RNN may have. Also, one of the methods used by the WA extraction method to generate the basis on which the Hankel matrices will be build, requires access to such a dataset. We created these datasets by passing the samples from the original datasets through the input vector to symbol mapping. This gives us a 'symbolized' version of the original datasets. Figure B.2 depicts this process.

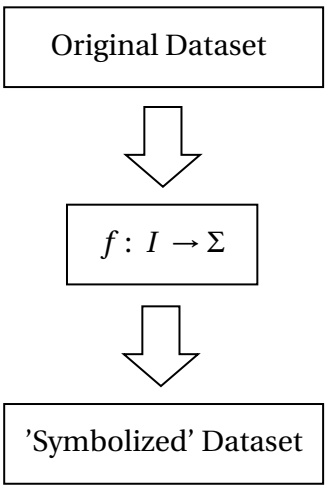


Figure B.2: Dataset conversion using input vector to symbol mapping

B.1.2. DISCRETIZATION IMPACT ANALYSIS

It is difficult to asses the overall affect of the input discretization just by looking at the input data itself as the RNN has learned a highly complex function between input and output and it is unclear how a certain reduction of input fidelity will affect the classification performance. Therefore we will assess the affect of the input discretization on the RNN's classification performance by evaluating the performance of the combined model on the 'symbolized' training set against that of the RNN on the original training set.

B.1.3. INPUT FEATURE DISTRIBUTION

Before attempting to map the input we inspected the input features themselves in more detail on the training set. We looked into the value range and distribution of each feature

to get a feel for the sort of information each feature carries. The idea was that perhaps some input features only had a very small contribution to the actual input and could be omitted completely without loosing to much information.

To investigate this hypothesis we computed value histograms for each input feature on the training set, see B.3. From these histograms is appears that feature 4 always has a constant value and does not carry any real information, features 1 and 7 to 12 only take on two values and features 2, 3, 5 and 6 have many different values. The histograms also show that the feature values are not normalized to a common value range. Some features range from -1.0 to 1.0 whilst others range from -1.0 to 6.0.

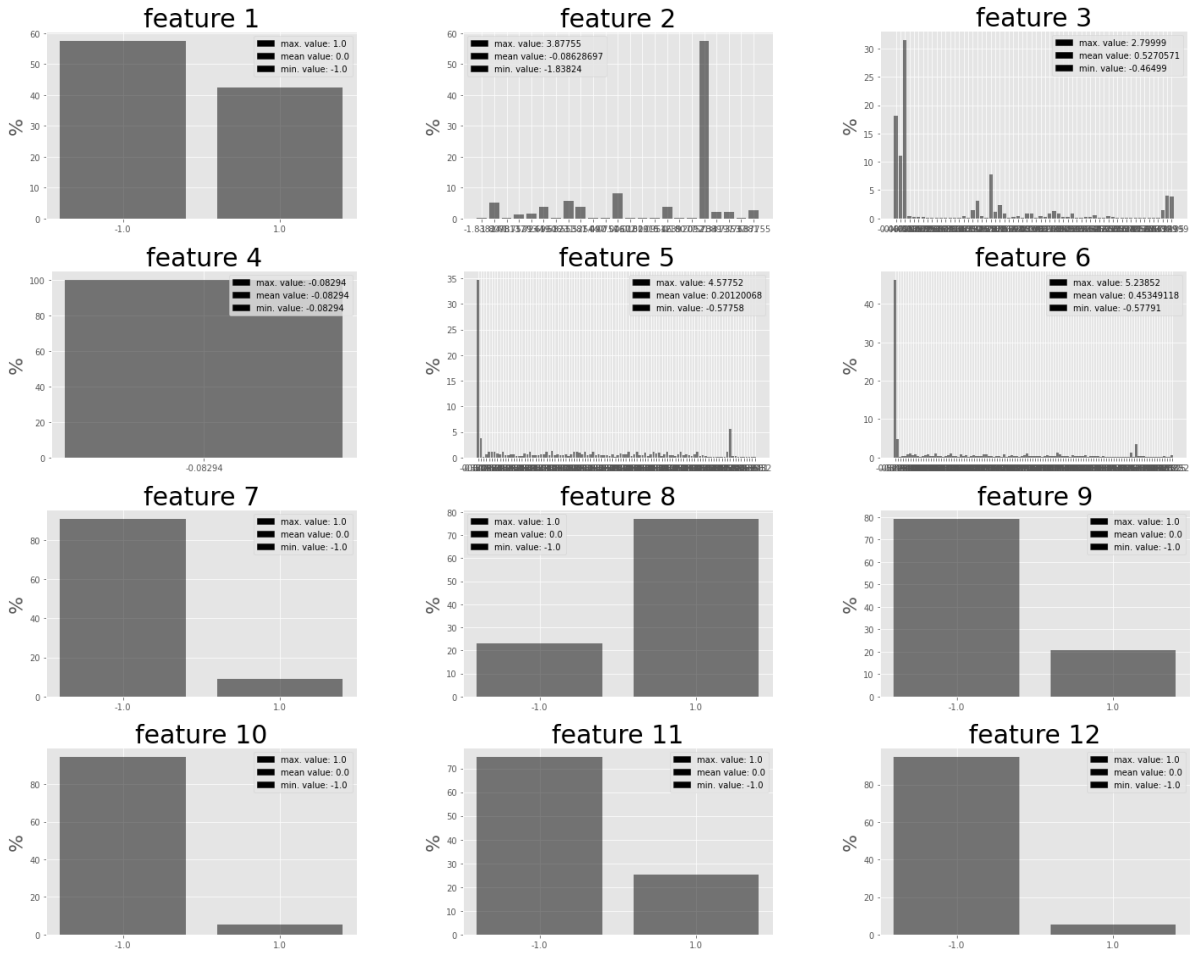


Figure B.3: Input feature histograms

We also conducted a Principal Component Analysis (PCA) on the input features. PCA transforms the original input feature vectors into vectors of new features that are linearly independent. If the original input features have any mutual correlation then the number of features can be reduced without loosing significant information. This would give a quick way to reduce the input space. The downside of this approach is that the new features are not directly linked to the input domain, i.e., their meaning or role in the input domain is not directly apparent. Loosing semantic information is not desirable as we will ultimately use the symbols from the constructed alphabet to try and explain the RNN's operation when interpreting the extracted automota. Although insightful, we decided not to use PCA further

in our input discretization for the aforementioned reasons.

B.1.4. HASHING

As a first experiment we made a mapping to a set of 100 symbols using a simple hashing approach with even weight across all input features. The advantage of this approach is that we do not need to infer the alphabet upfront from the input domain which would require access to, for example, the RNN's training set. We can simply add symbols to the alphabet when we encounter input values that have not yet been mapped. This means the approach could be used in an online setting in which the alphabet would be inferred on the fly.

A first naive attempt was simply reducing the precision of each feature by rounding them to a certain number of decimals. For each input vector the string representation of these rounded values were then combined into a string representation of the vector. This string representation of the vector was then used a hash key in a symbol set to convert it into a symbol. Clearly such a mapping cannot be easily inverted, but it would give us a feeling of what alphabet size could roughly be achieved in this way. It quickly became apparent that even when reducing the precision of the individual feature values to 1 decimal the resulting alphabet size was still very large at 1177 symbols.

The downside of using a string hash as the set key was that we could not control the alphabet size directly, but only by reducing the feature precision. We experimented further with a variation of this idea in which we calculated the hash value of a vector of scaled down feature values directly. Using modulo arithmetic on the absolute value of this hash allowed us to directly control the number of unique symbols that was generated. Using this method we were able to produce an alphabet of 100 symbols.

To create the inverse mapping for this hashing approach we can use the mean values of the individual feature values over all the vectors that where mapped to each symbol. Its obvious that we need a fair amount of observations before we can calculate steady mean values. In an online setting these mean values could shift over time as more an more different input vectors are mapped to their corresponding symbols. To evaluate our hashing approach we have chosen to use the training set to calculate the mean values upfront so we have a consistent mapping during our experiments.

Clearly the downside of this approach is that we cannot control how the different input vectors will collide and thus how well the original information will be retained. The chosen feature precision will influence the hashing values and thus how many different input samples will be mapped to the same symbol. Without knowing upfront how the feature values are distributed it may be very difficult to find the appropriate settings. Also this method treats all features equally and the hashing process does not take value 'nearness' into account, but rather treats each sample as an independent value. Other more sophisticated methods could perhaps group vectors that are close to each other together and perform a non linear separation of the input domain, preserving more of the original information.

Using the input vector to symbol mapping and the inverse mapping from symbol to vector mean values in conjunction allowed us to evaluate the impact of these mappings on the RNN's classification performance. When we applied this mapping to the setup of figure 6.1 and evaluated the performance of the combination on the RNN's training set we found very poor performance. The RNN with input adapter achieved an accuracy of just 72%. This is

much lower than the 99% of the RNN when applied directly to the training set. A drop in performance was to be expected, but when we look into the classification report details of table B.1 we see the F1-score on the negative class is extremely poor. It is clear this setup will not be usable for our experiments.

	precision	**recall**	**f1-score**	**support**
0	0.18	0.04	0.07	3981
1	0.75	0.94	0.83	12317
accuracy			0.72	16298
macro avg	0.46	0.49	0.45	16298
weighted avg	0.61	0.72	0.65	16298

Table B.1: RNN classification report on training set using hashing based input discretization

Clearly our discretization is removing to much relevant information from the input, causing the RNN to misclassify samples. As we felt that an alphabet of 100 symbols was already quite large with respect the comprehensibility of the resulting automata we did not want to increase the alphabet size.

B.1.5. CLUSTERING

After the unsuccessful hashing approach to the discretization problem we looked into other solutions. We ultimately decided to employ a clustering algorithm called k-means. K-means is an unsupervised learning algorithm that can divide input data into a number of distinct clusters. The algorithm will cluster input vectors around the nearest mean values in such a way as to minimize the within cluster variance. This algorithm is used in many scenario's where data needs to be separated into distinct bins or clusters and is widely applied in the literature [Jai10].

We used the k-means class from the Scikit learn Python library [sci20] to perform the k-means clustering on the datasets. The benefit of using this implementation is that it also provides an inverse transform which is precisely what we need. After converting the datasets using the k-means clustering and its inverse transform we evaluated the effect of the discritization.

We fit the k-means model on the training set and apply it to the other datasets. This way our adapter is usable in a setting where we do not know the data upfront, other than the original training data. This is a necessary requirement as the rule extraction approaches can generate input samples during extraction that may represent previously unseen data. Also if we would fit the k-means model on each dataset separately we cannot guarantee consistency in the feature to symbol mapping across the datasets. This could lead to reduced classification performance as well as interpretation issues when we need to map symbols back to input features when we analyze the extracted automata.

When we applied this mapping to the setup of figure 6.1 and evaluated the performance of the combination on the RNN's training set we found excellent performance. The RNN with k-means input adapter achieved an accuracy of 98%. This only marginally lower than the 99% of the RNN when applied directly to the training set. We expected a larger drop in performance, but even when we look into the classification report details of table B.2 we

see an excellent F1-score of 95% for the negative class and 99% for the positive class.

	precision	recall	f1-score	support
0	0.98	0.92	0.95	5513
1	0.99	1.00	0.99	27573
accuracy			0.98	33086
macro avg	0.98	0.96	0.97	33086
weighted avg	0.98	0.98	0.98	33086

Table B.2: RNN classification report on training set using K-means based input discretization

	precision	recall	f1-score	support
0	0.99	0.45	0.62	10225
1	0.60	1.00	0.75	8438
accuracy			0.70	18663
macro avg	0.80	0.72	0.68	18663
weighted avg	0.81	0.70	0.68	18663

Table B.3: RNN classification report on test set using K-means based input discretization

When we evaluated the *k*-means input adapter on the test set, however, we did see a significant drop in performance. Now the accuracy is down to 70%. When we look into the classification report details of table B.3 we also see significantly lower F1-scores for the negative and positive classes.

We initially applied the *k*-means transformation on the original input features and performed min/max scaling only afterwards on the inversely transformed data. To see what the effect of upfront scaling would be on the *k*-means transformation we also performed an experimented with this setup. Table B.4 shows the result of this experiment. Although the performance on the test set has improved, we still see a significant decrease in the combined model's ability to detect the negative class as expressed by the low recall on the negative class and low precision on the positive class.

	precision	recall	f1-score	support
0	0.99	0.63	0.77	10225
1	0.69	1.00	0.82	8438
accuracy			0.80	18663
macro avg	0.84	0.81	0.79	18663
weighted avg	0.86	0.80	0.79	18663

Table B.4: RNN classification report on test set using K-means based input discretization with min/max scaling.

The difference in performance on the training and test set can probably be explained by the fact that the *k*-means clustering model was fitted on the training set and than applied to the validation and test sets. If the data is distributed differently between the datasets this may cause a poor fit of the validation and test sets to the cluster centers inferred on the training set. To see if this may be the case we looked into the sum of distances of the samples in each dataset to their closest cluster center as inferred on the training set. This

measure is known as the *k*-means score and is in affect the error measure which the *k*-means algorithm attempts to minimise during fitting. Table B.5 shows this score value for the different datasets. We can clearly see that indeed the validation and test set have much larger values than the training set.

	***k*-means score**
training set	-22602.86
validation set	-51963.46
test set	-119839.58

Table B.5: *k*-means score on the different datasets

This makes the *k*-means input adapter unsuitable for our experiments as the rule extraction algorithms could generate samples during extraction that may not yet have been encountered which may lead to poor translation into the RNN's input domain. This distortion may influence the performance of the extraction process itself and the reliability of the results. For this reason we have decided not to proceed with this approach.

In turns out that data discretization is a challenging task, however, it is also an important task that has great impact on all subsequent steps. There are many different discretization algorithms and approaches described in the literature [Liu+02; Jai10] which might perform better, but time does not permit us to further investigate this aspect as it is not the main focus of our research.

B.2. RNN (RE-)TRAINING ON SYMBOLS

After these experiments we concluded that the approach with a separate input adapter would not lead to usable results. We then tried another approach in which we re-implemented the RNN and changed the input layer to accept the symbols inferred by the *k*-means approach directly. Of course we would like an extraction approach where we can use an RNN as-is, but it seems that applying a separate adapter is not a viable option, at least not for our botnet RNN.

The idea behind changing the RNN to accept the symbols directly and re-training it on the 'symbolized' training set is that this way the RNN is trained on the discretized data directly which may lead to better classification performance as the network can adjust itself to the different input domain.

Initially we attempted to use the symbol indexes directly as a single input feature, but this was unsuccessful as the RNN failed to train properly. We then used a one-hot encoding on the symbol indexes, which is a commonly used approach in machine learning when dealing with categorical input features [GB16]. This encoding translates a symbol index i into an array with a length equal to the number of symbols, in which the ith cell has a value of one and the remaining cells are zero. This encoded data transforms the single input feature into, in our case, 100 orthogonal input features. This allows the RNN to adjust all 100 input weights individually to best fit the input data.

We successfully trained this RNN setup on the 'symbolized' training set to an accuracy of 99%. We then evaluated the trained RNN on the 'symbolized' test set. Table B.6 shows the classification report for this evaluation.

	precision	recall	f1-score	support
0	0.77	0.37	0.50	10225
1	0.53	0.87	0.66	8438
accuracy			0.60	18663
macro avg	0.65	0.62	0.58	18663
weighted avg	0.66	0.60	0.57	18663

Table B.6: RNN classification report on 'symbolized' test set using one-hot encoding.

As we can see, the RNN does not perform well on the test set. Especially the recall on the negative class and the precision on the positive class are quite poor. From these results we concluded that this setup could not be used for our experiments.

We then experimented with using an embedding layer which turns the symbol indexes into dense vectors of fixed size. An embedding layer will find similarities between indexes by looking at their context, i.e., the other indexes in a input sequence. The vectors of each embedding get updated during training to express the relationships between indexes. The idea of using this layer is that this additional information on symbol relationship may help the RNN to generalize better and improve its classification performance on unseen data. Although an interesting theory the addition of an embedding layer did not significantly improve the training and classification performance of the RNN compared to the one-hot encoding setup.

B.2.1. REDISTRIBUTION OF DATASETS

From these experiments it became apparent that training the RNN on the converted versions of the dataset used by Poon proved difficult. Of course our 'symbolizing' of the data using the k-means approach describe in the previous paragraph will be a contributing factor. However, another cause may be the different class distributions in the datasets. The training and validation set have around 82% positives samples, while the test set has around 45% positive samples. This difference may cause the RNN to generalize poorly on the negative class. Also the fact that the k-means model was fit on the training set and then applied as-is to the other datasets may have caused differences between the datasets which the RNN model failed to generalize on.

To test this hypothesis we redistributed the samples in the datasets to form three new datasets. We combined the original datasets and randomly re-sampled them to form the new training, validation and test set. This way we have a more even distribution of negative samples in the training set and the affect of our k-means discretization approach is 'spread' across all datasets, which may help the RNN to generalize better on the new input domain. When we trained both versions of the RNN using this new training set and evaluated them on the new test set we did indeed noticed a significant increase in performance. The one-hot encoding version and embedding version of the RNN both trained to an accuracy of around 97%. Table B.7 and B.8 show the classification report on the test set for the one-hot encoding and embedding approach respectively.

We can clearly see that the performance of both approaches significantly improved and that both approaches perform almost identical. After these results we felt confident we

	precision	recall	f1-score	support
0	0.96	0.96	0.96	6633
1	0.98	0.98	0.98	12030
accuracy			0.97	18663
macro avg	0.97	0.97	0.97	18663
weighted avg	0.97	0.97	0.97	18663

Table B.7: RNN classification report on redistributed 'symbolized' test set using on-hot encoding

	precision	recall	f1-score	support
0	0.96	0.94	0.95	6633
1	0.97	0.98	0.97	12030
accuracy			0.96	18663
macro avg	0.96	0.96	0.96	18663
weighted avg	0.96	0.96	0.96	18663

Table B.8: RNN classification report on redistributed 'symbolized' test set using embedding

could use this RNN in the rest of our research. We ultimately decided to use the one-hot encoding method for the remainder of our research as it felt simpler, but we could have used either method.

C

RULE EXTRACTION EXPERIMENTS

In this appendix information can be found on the case study rule extraction experiments that was too large or detailed to include in the main content.

C.1. SLOW PREDICTION AND HIGH MEMORY CONSUMPTION FOR CASE STUDY RNN

The rule extractions proved extremely slow on our Keras based RNN. The extraction steps took around 50-100 times slower than with the Dynet based RNN implementation from Weis et al. , making a direct comparison between the RNN implementations impossible.

When we looked at the single prediction time of our Keras based RNN this seemed to be very poor. A single prediction took in the region of 0.6 seconds. It has to be noted that, for the reason mentioned in A, we could not use GPU acceleration for our RNN model. This may have negatively impacted the prediction time performance of the Keras based RNN, however, it does not fully explain the extremely poor performance. We did have several issues with the version of Tensorflow (version 2.1) we used and there seems to be a general sense in the machine learning community that there is a degradation in performance with respect to previous versions [1].

As our Keras based RNN had slow single word prediction, we changed the part of the WA extraction method code that builds the Hankel matrices to retrieve the RNN prediction values in batch for all pre- and suffixes, rather than per individual pre- and suffix. This greatly improved the speed in which the Hankel matrices were constructed. However, during the extractions we experienced out-of-memory exceptions with the Keras based RNN which we could trace back to this change. All the combinations of pre- and suffixes were now passed to the prediction code at the same time. Before the data could be used by the RNN it had to be padded to length and one-hot encoded, which resulted in a large amount of data that had to be put in memory. We improved the situation by using a data generator for the predictions which processes the data in smaller batches.

For larger basis sizes the Dynet based RNN version based from Weis et al. performed

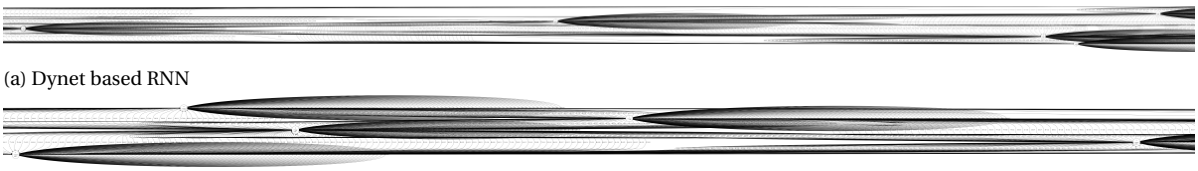
[1] see for example https://github.com/keras-team/keras/issues/13118

worse than the Keras based version in terms om prediction time. The implementation from Weis et al. can only perform single predictions and this may explain the slow performance on large batches as it effectively processes them one by one.

The DFA extraction algorithm will query the RNN frequently during the exact learning process. Either to answer the membership queries or to refine the abstraction of the RNN that is used to answer the equivalence queries. Especially the refinement process may use a large number of samples to determine the RNN's state clustering (the macrostates) from which the abstraction is build, depending on the state space of the RNN. This means that poor prediction time performance of the RNN will potentially be magnified by orders of magnitude, leading to very slow extraction. Because of the relative complexity of the DFA extraction code we did not feel comfortable to try and change the code to also use batches, rather than repeated single predictions.

C.2. OVERALL BEST PERFORMING AUTOMATA

Figure C.1 and C.2 show cropped versions of the overall best performing automata that were extracted in the case study extraction experiments. The actual automata were too large to include in this document.

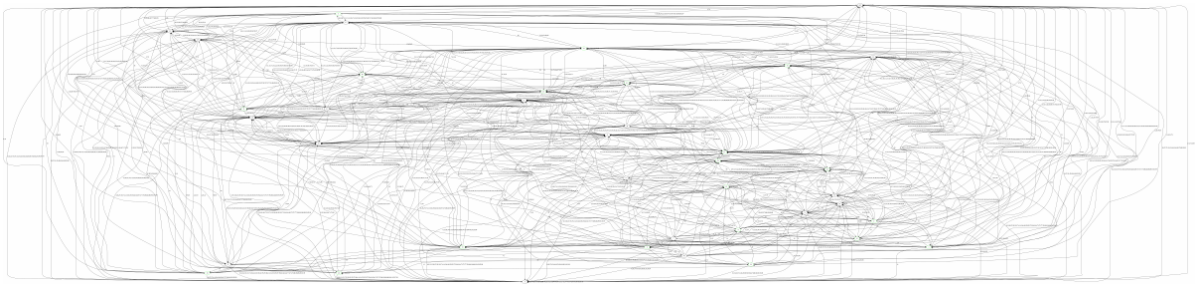


(a) Dynet based RNN

(b) Keras based RNN

Figure C.1: Overall best performing WA extracted.

(a) Dynet based RNN



(b) Keras based RNN

Figure C.2: Overall best performing DFA extracted.

D

RULE EXTRACTION ON 'NOISY' MODELS

During our research we performed several experiments to try and answer our original research questions. Over the course of these experiments it became clear that the DFA extraction method is affected by noise or 'non-DFA' behavior, a fact that is also pointed out by its researchers in [WGY18a].

To further investigate this issue we performed additional experiments. With these experiments we hope to gain further insight into each method's robustness with respect to 'noisy' data. We want to explore how each method can be applied to 'noisy' models and to what extend this affects their performance. Also, it will be interesting to see if the methods can be tuned to approximate the 'noisy' models to a certain degree even if they cannot exactly match them. In other words, we want to explore if the methods allow to control the granularity of the extracted automata through their hyper-parameters or execution time.

We feel these aspects play an important role in the practical applicability of each rule extraction method as real life RNN models will seldom exhibit pure DFA behaviour.

D.1. SETUP

For these experiments we compared both rule extraction methods on datasets generated from selected Tomita grammars in which we randomly changed the classification label for a particular percentage of samples. These datasets no longer represent the Tomita grammar exactly, but have added 'noisy' or 'non-DFA' behaviour. We selected Tomita grammars 1 and 7 for these experiments as a RNN can easily be learned for these grammars and they form a good range of complexities and DFA sizes. Furthermore, as we have seen in the experiments of chapter 7 that these grammars represent a wide range of extraction behaviour, at least for the WA extraction method. To also asses extraction on 'real' non-DFA models we added a non-regular grammar to the original seven Tomita grammars. Table D.1 describes this non-regular grammar.

We used the RNN implementation from Weiss et al. for these experiments. During initial experiments we noticed that it was easier to train RNNs based on the implementation from Weiss et al. than it was to train RNNs based on our Keras implementation. This most likely has to do with how the training code from Weiss et al. actually trains the RNN. It performs

G	description
-1	$L0 = \{w \in \{0,1\}^* : w = 0^i 1^i\}$, for some $i \geq 0$

Table D.1: Description of the additional non-regular grammar model.

several training steps using batches of equal word length, random batches and one large single batch to train the RNN. This approach is most likely better equipped to efficiently learn Tomita grammar like models than simply splitting the dataset in multiple batches like we did for the Keras based model. We used a RNN configuration with a single LSTM layer with 12 neurons.

We trained a RNN to 100% on the non-DFA grammar using a training set generated on that grammar with words lengths up to a 100 characters. This way we can make sure the RNN is properly trained on the grammar model and that it can correctly recognise all words that are presented to it during extraction, even if they become large. This is important as especially the WA method may generate quite large words when larger Hankel basis sizes are used. The test set for these experiments contained words lengths of up to a 50 characters. The trained RNN also reached an accuracy of 100% on the test set.

For the experiments in which we added Gaussian noise to the datasets, however, we could not train a RNN on the 'noisy' Tomita grammars in a timely fashion when we used a training set with words lengths up to a 100 characters. We had to reduce the string lengths to a maximum of 50 characters in order to train the RNN in a reasonable time frame. We trained the RNN's to 100% accuracy on the modified training sets. In these experiments we also used the training set as test set. As the changes to the class labels in each dataset are completely random it is highly unlikely that a model trained on the training set will achieve good classification performance on the separately modified test set.

We experimented with 0, 2 and 5% added noise. For the WA extraction method we used the minimal DFA's sizes of the Tomita grammars as rank value, similar to what we did in chapter 7. For the non-DFA model we used rank values between 2 and 6, as we felt this would be sufficient. Furthermore we used sampling from the uniform distribution on symbols with a maximum length between 15 and 25 and Hankel basis sizes between 50 and 400. We felt these values would be sufficient and would most likely lead to word lengths in the Hankel basis generation that fall inside the range on the RNN's training data. For the DFA extraction method we used extraction timeout increments of 10 seconds starting with 1 second up to 100 seconds. We feel these settings for the rule extraction method hyperparameters cover the most appropriate range of settings and should allow the methods to perform sufficiently well.

Because some of the grammar models we evaluated are highly imbalanced, like Tomita 1 for which the positive class only accounts for 3% of all data, we measured the minimum fidelity figures so any problem with the minority class is highlighted.

D.2. EXPERIMENT AND RESULTS

For the non-regular grammar the DFA extraction initially terminated prematurely on a automaton that was too small to properly represent the trained RNN. We therefore had to increase the initial split depth parameter to prevent this. This value cannot be set too large

because this makes the required extraction time infeasible. We found, however, that in the experiments on the non-regular grammar the DFA extraction method would consistently exit prematurely on a DFA with 26 states, regardless of how high we set the initial split depth. For the experiments on the 'noisy' Tomita grammar models we settled for a initial split depth value of 20, since this was large enough to extract DFA's with around 30 states or more.

For the experiments on the non-regular grammar model the minimum fidelity of the best performing automata extracted by the WA extraction method was 96% for a rank value of 6 and a Hankel basis size of 400. As mentioned, the DFA method extracted a DFA with a maximum of 26 states for time limits of 27 seconds or more, which reached a minimum fidelity of 100%.

We found that for the experiments on the 'noisy' version of the Tomita grammars the WA method would not extract WAs with a minimum fidelity of more than 53% on the modified train set. When we evaluated the minimum fidelity on the unmodified test set we found that some WAs achieved near perfect fidelity values of 98% or better. It seems that the WA method failed to 'learn' the added 'noisy' behaviour in these cases and extracted an automaton that models the original Tomita grammar.

For the DFA method we found that it would extract very large automata on the 'noisy' version of the Tomita grammars, even for small extraction time limits. The automata quickly reached a sizes of 95 states or more, but still had quit poor fidelity figures. For a time limit of just 15 seconds, the extraction on the 'noisy' version of the Tomita 1 produced a DFA of 481 states that had a minimum fidelity of 43%.

In all the experiments we found that the WA extraction method produced automata with widely varying fidelities between runs, similar to what we have seen in our other experiments.

D.3. ANALYSIS

As was to be expected the DFA method cannot handle the 'noisy' models very well. For the 'noisy' Tomita models it generated extremely large automata that failed to approximate the target model. For the non-regular grammar it was able to extract an automaton that reached 100% fidelity on the test set. However, when we further investigated this we found that, for longer sequences than the maximum of 50 symbols in the test set, this quickly dropped. This indicates that the DFA approximated the model up to words of 50 characters, but not for longer sequences.

The WA method seems less affected by 'noisy' or non-DFA models. Although the fidelity figures obtained in the experiments were not very good, we feel this has more to do with the evaluated range of Hankel basis sizes than the method itself.

The reason for this difference in behaviour, we believe, is that the WA method can take a purely numerical approach to approximate the target model in the form of the produced weighted automata. The DFA method, however, can only try to add more states and transitions to enhance the automaton's fidelity. It has to uniquely encode a path in the automaton for each non-regular pattern in the target model. This means that the DFA method will struggle when the target model contains large amount of non-DFA behaviour.

We have also seen that when using the DFA extraction method on these non-regular models the initial split depth parameter needs to be taken into account as well to prevent the algorithm to prematurely terminate on small DFA's. All in all it is clear that the DFA method cannot really be used on pure non-DFA models, especially on longer sequences, as this would require extremely large DFA's.

BIBLIOGRAPHY

SCIENTIFIC PEER-REVIEWED ARTICLES

[ADT95] Robert Andrews, Joachim Diederich, and Alan B Tickle. "Survey and critique of techniques for extracting rules from trained artificial neural networks". In: *Knowledge-based systems* 8.6 (1995), pp. 373–389.

[AEG18] Stéphane Ayache, Rémi Eyraud, and Noé Goudian. "Explaining Black Boxes on Sequential Data using Weighted Automata". In: *Proceedings of the 14th International Conference on Grammatical Inference*. 2018.

[Ang87] Dana Angluin. "Learning regular sets from queries and counterexamples". In: *Information and computation* 75.2 (1987), pp. 87–106.

[Arr+17] Leila Arras, Grégoire Montavon, Klaus-Robert Müller, and Wojciech Samek. "Explaining Recurrent Neural Network Predictions in Sentiment Analysis". In: *Proceedings of the 8th Workshop on Computational Approaches to Subjectivity, Sentiment and Social Media Analysis*. 2017, pp. 159–168.

[Bal+14] Borja Balle, Xavier Carreras, Franco M Luque, and Ariadna Quattoni. "Spectral learning of weighted automata". In: *Machine learning* 96.1-2 (2014), pp. 33–63.

[BDR09] Raphaël Bailly, François Denis, and Liva Ralaivola. "Grammatical inference as a principal component analysis problem". In: *Proceedings of the 26th Annual International Conference on Machine Learning*. ACM. 2009, pp. 33–40.

[BGW00] Adam L Buchsbaum, Raffaele Giancarlo, and Jeffery R Westbrook. "On the determinization of weighted finite automata". In: *SIAM Journal on Computing* 30.5 (2000), pp. 1502–1531.

[BM18] Borja Balle and Mehryar Mohri. "Generalization bounds for learning weighted automata". In: *Theoretical Computer Science* 716 (2018), pp. 89–106.

[BSF+94] Yoshua Bengio, Patrice Simard, Paolo Frasconi, et al. "Learning long-term dependencies with gradient descent is difficult". In: *IEEE transactions on neural networks* 5.2 (1994), pp. 157–166.

[CBP19] Manomita Chakraborty, Saroj Kumar Biswas, and Biswajit Purkayastha. "Rule Extraction from Neural Network Using Input Data Ranges Recursively". In: *New Generation Computing* 37.1 (2019), pp. 67–96.

[CP71] Jack W. Carlyle and Azaria Paz. "Realizations by stochastic finite automata". In: *Journal of Computer and System Sciences* 5.1 (1971), pp. 26–40.

[CSM89] Axel Cleeremans, David Servan-Schreiber, and James L McClelland. "Finite state automata and simple recurrent networks". In: *Neural computation* 1.3 (1989), pp. 372–381.

[DG07] Manfred Droste and Paul Gastin. “Weighted automata and weighted logics”. In: *Theoretical Computer Science* 380.1-2 (2007), pp. 69–86.

[EL06] Terence A Etchells and Paulo JG Lisboa. “Orthogonal search-based rule extraction (OSRE) for trained neural networks: a practical and efficient approach”. In: *IEEE transactions on neural networks* 17.2 (2006), pp. 374–384.

[FM15] Enric Junque de Fortuny and David Martens. “Active learning-based pedagogical rule extraction”. In: *IEEE transactions on neural networks and learning systems* 26.11 (2015), pp. 2664–2677.

[GB16] Cheng Guo and Felix Berkhahn. “Entity embeddings of categorical variables”. In: *arXiv preprint arXiv:1604.06737* (2016).

[GCB16] Ian Goodfellow, Aaron Courville, and Yoshua Bengio. “Deep learning”. In: (2016).

[Gil+92] C Lee Giles, Clifford B Miller, Dong Chen, Hsing-Hen Chen, Guo-Zheng Sun, and Yee-Chun Lee. “Learning and extracting finite state automata with second-order recurrent neural networks”. In: *Neural Computation* 4.3 (1992), pp. 393–405.

[GK95] Sally A Goldman and Michael J Kearns. “On the complexity of teaching”. In: *Journal of Computer and System Sciences* 50.1 (1995), pp. 20–31.

[GMH13] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. “Speech recognition with deep recurrent neural networks”. In: *2013 IEEE international conference on acoustics, speech and signal processing*. IEEE. 2013, pp. 6645–6649.

[Gui+18] Riccardo Guidotti, Anna Monreale, Salvatore Ruggieri, Franco Turini, Fosca Giannotti, and Dino Pedreschi. “A survey of methods for explaining black box models”. In: *ACM computing surveys (CSUR)* 51.5 (2018), p. 93.

[Haw04] Douglas M Hawkins. “The problem of overfitting”. In: *Journal of chemical information and computer sciences* 44.1 (2004), pp. 1–12.

[HH94] Bill G Horne and Don R Hush. “Bounds on the complexity of recurrent neural network implementations of finite state machines”. In: *Advances in neural information processing systems*. 1994, pp. 359–366.

[HKZ12] Daniel Hsu, Sham M Kakade, and Tong Zhang. “A spectral algorithm for learning hidden Markov models”. In: *Journal of Computer and System Sciences* 78.5 (2012), pp. 1460–1480.

[IS15] Sergey Ioffe and Christian Szegedy. “Batch normalization: Accelerating deep network training by reducing internal covariate shift”. In: *arXiv preprint arXiv:1502.03167* (2015).

[Jac05] Henrik Jacobsson. “Rule extraction from recurrent neural networks: Ataxonomy and review”. In: *Neural Computation* 17.6 (2005), pp. 1223–1263.

[Jai10] Anil K Jain. “Data clustering: 50 years beyond K-means”. In: *Pattern recognition letters* 31.8 (2010), pp. 651–666.

[JM15] Michael I Jordan and Tom M Mitchell. “Machine learning: Trends, perspectives, and prospects”. In: *Science* 349.6245 (2015), pp. 255–260.

[JMM96] Anil K Jain, Jianchang Mao, and KM Mohiuddin. "Artificial neural networks: A tutorial". In: *Computer* 3 (1996), pp. 31–44.

[Joh80] Philip N Johnson-Laird. "Mental models in cognitive science". In: *Cognitive science* 4.1 (1980), pp. 71–115.

[KM09] Eyal Kolman and Michael Margaliot. "Extracting symbolic knowledge from recurrent neural networks—a fuzzy logic approach". In: *Fuzzy Sets and Systems* 160.2 (2009), pp. 145–161.

[Kon01] Igor Kononenko. "Machine learning for medical diagnosis: history, state of the art and perspective". In: *Artificial Intelligence in medicine* 23.1 (2001), pp. 89–109.

[LBH15] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. "Deep learning". In: *nature* 521.7553 (2015), p. 436.

[Liu+02] Huan Liu, Farhad Hussain, Chew Lim Tan, and Manoranjan Dash. "Discretization: An enabling technique". In: *Data mining and knowledge discovery* 6.4 (2002), pp. 393–423.

[Mik12] Tomáš Mikolov. "Statistical language models based on neural networks". In: *Presentation at Google, Mountain View, 2nd April* 80 (2012).

[MJ01] Larry R Medsker and LC Jain. "Recurrent neural networks". In: *Design and Applications* 5 (2001).

[Mon+17] Grégoire Montavon, Sebastian Lapuschkin, Alexander Binder, Wojciech Samek, and Klaus-Robert Müller. "Explaining nonlinear classification decisions with deep taylor decomposition". In: *Pattern Recognition* 65 (2017), pp. 211–222.

[MP43] Warren S McCulloch and Walter Pitts. "A logical calculus of the ideas immanent in nervous activity". In: *The bulletin of mathematical biophysics* 5.4 (1943), pp. 115–133.

[MSM18] Grégoire Montavon, Wojciech Samek, and Klaus-Robert Müller. "Methods for interpreting and understanding deep neural networks". In: *Digital Signal Processing* 73 (2018), pp. 1–15.

[OG96] Christian W Omlin and C Lee Giles. "Extraction of rules from discrete-time recurrent neural networks". In: *Neural networks* 9.1 (1996), pp. 41–52.

[Pas+14] Razvan Pascanu, Caglar Gulcehre, Kyunghyun Cho, and Yoshua Bengio. "How to construct deep recurrent neural networks". In: *Proceedings of the Second International Conference on Learning Representations (ICLR 2014)*. 2014.

[Pow11] David Martin Powers. "Evaluation: from precision, recall and F-measure to ROC, informedness, markedness and correlation". In: (2011).

[Ros58] Frank Rosenblatt. "The perceptron: a probabilistic model for information storage and organization in the brain." In: *Psychological review* 65.6 (1958), p. 386.

[SP97] Mike Schuster and Kuldip K Paliwal. "Bidirectional recurrent neural networks". In: *IEEE Transactions on Signal Processing* 45.11 (1997), pp. 2673–2681.

[Tom82] Masaru Tomita. "Dynamic construction of finite-state automata from examples using hill-climbing." In: *Proceedings of the Fourth Annual Conference of the Cognitive Science Society*. 1982, pp. 105–108.

[VO04] A Vahed and Christian W Omlin. “A machine learning method for extracting symbolic knowledge from recurrent neural networks”. In: *Neural Computation* 16.1 (2004), pp. 59–71.

[Wan+18b] Qinglong Wang, Kaixuan Zhang, Alexander G Ororbia II, Xinyu Xing, Xue Liu, and C Lee Giles. “An empirical evaluation of rule extraction from recurrent neural networks”. In: *Neural computation* 30.9 (2018), pp. 2568–2591.

[WGY18a] Gail Weiss, Yoav Goldberg, and Eran Yahav. “Extracting Automata from Recurrent Neural Networks Using Queries and Counterexamples”. In: *Proceedings of the 35th International Conference on Machine Learning*. 2018, pp. 5247–5256.

SCIENTIFIC NON PEER-REVIEWED ARTICLES

[Sal+18] Hojjat Salehinejad, Julianne Baarbe, Sharan Sankar, Joseph Barfett, Errol Colak, and Shahrokh Valaee. *Recent Advances in Recurrent Neural Networks*. 2018.

[SJ16] Rohollah Soltani and Hui Jiang. *Higher order recurrent neural networks*. 2016.

[SWM17] Wojciech Samek, Thomas Wiegand, and Klaus-Robert Müller. *Explainable artificial intelligence: Understanding, visualizing and interpreting deep learning models*. 2017.

[Wan+18a] Qinglong Wang, Kaixuan Zhang, Xue Liu, and C Lee Giles. *Verification of Recurrent Neural Networks Through Rule Extraction*. 2018.

[Wan+18c] Qinglong Wang, Kaixuan Zhang, II Ororbia, G Alexander, Xinyu Xing, Xue Liu, and C Lee Giles. *A Comparative Study of Rule Extraction for Recurrent Neural Networks*. 2018.

BOOKS

[Bou98] Nicolas Bourbaki. *Algebra I: chapters 1-3*. Vol. 1. Springer Science & Business Media, 1998.

[DKV09] Manfred Droste, Werner Kuich, and Heiko Vogler. *Handbook of weighted automata*. Springer Science & Business Media, 2009.

[Hay+09] Simon S Haykin et al. *Neural networks and learning machines*. New York: Prentice Hall, 2009.

[Hop08] John E Hopcroft. *Introduction to automata theory, languages, and computation*. Pearson Education India, 2008.

[Mit06] Tom Michael Mitchell. *The discipline of machine learning*. Vol. 9. Carnegie Mellon University, School of Computer Science, Machine Learning, 2006.

[Sak09] Jacques Sakarovitch. *Rational and recognisable power series*. Springer, 2009, pp. 105–174.

MASTER THESES

[Poo18] Tho Poon. “Botnet detection using recurrent neural networks”. 2018.

LINKS

[Goo20] Google. *Google Colaboratory*. 2020. URL: https://colab.research.google.com/notebooks/welcome.ipynb (visited on 03/05/2020).

[jup20] jupyter.org. *Jupyter Notebook*. 2020. URL: https://jupyter.org/ (visited on 03/05/2020).

[ker20] keras.io. *Keras: The Python Deep Learning library*. 2020. URL: https://keras.io/ (visited on 03/05/2020).

[sci20] scikit-learn.org. *scikit-learn, Machine Learning in Python*. 2020. URL: https://scikit-learn.org/ (visited on 03/05/2020).

[WGY18b] Gail Weiss, Yoav Goldberg, and Eran Yahav. *Google Colaboratory notebook DFA rule extraction method*. 2018. URL: https://colab.research.google.com/drive/1tkJK1rJVEg9e-QcWOxErDb3cQq9UR-yR#scrollTo=wRIcWvcLTFdr (visited on 03/05/2020).